

---

# Ceph documentation

*Release dev*

**Ceph developers**

August 08, 2012



# CONTENTS

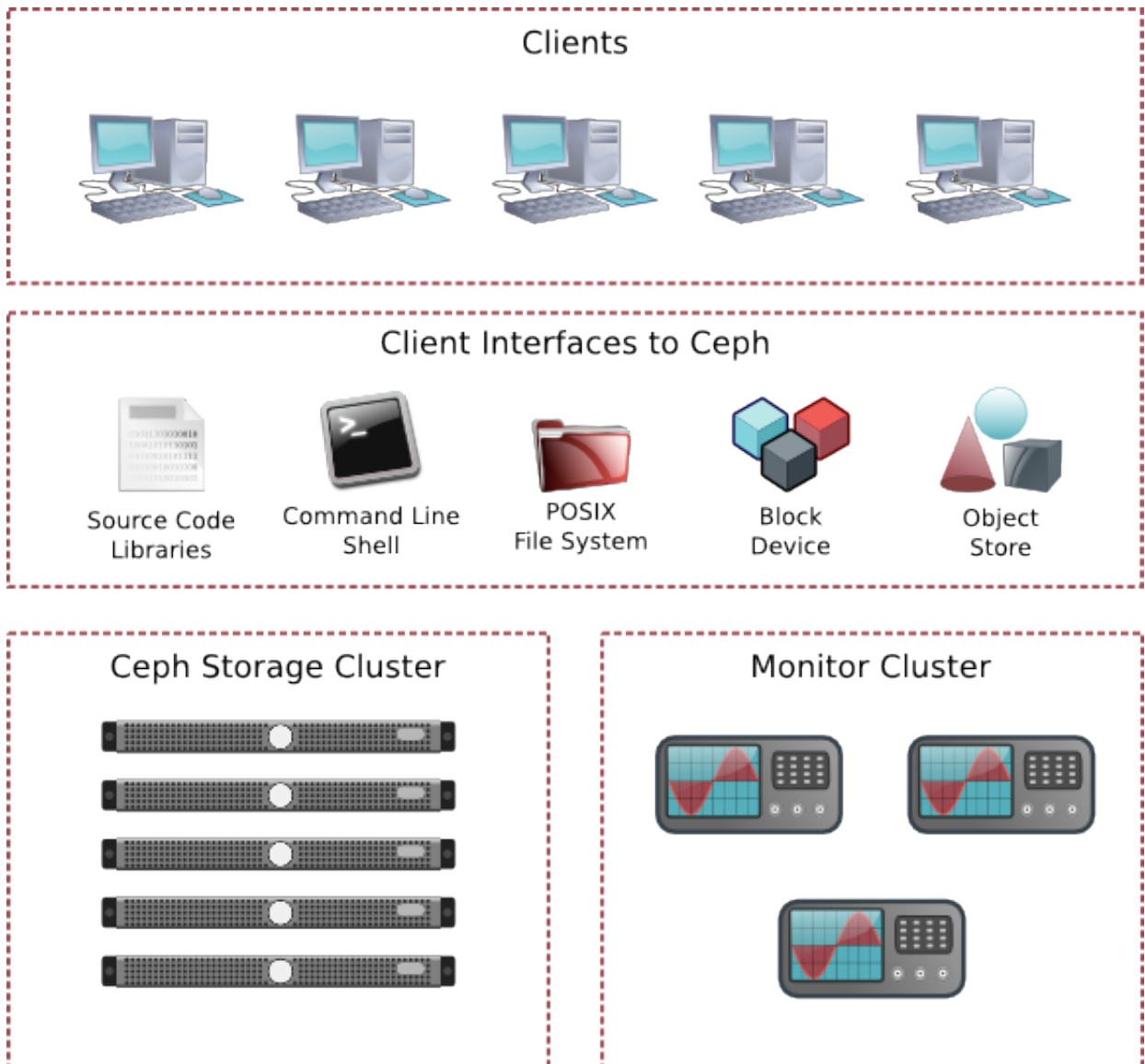
|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Getting Started</b>                              | <b>3</b>  |
| 1.1      | 5-minute Quick Start . . . . .                      | 3         |
| 1.2      | RBD Quick Start . . . . .                           | 4         |
| 1.3      | Ceph FS Quick Start . . . . .                       | 5         |
| 1.4      | Get Involved in the Ceph Community! . . . . .       | 5         |
| 1.5      | Installing Ceph Manually . . . . .                  | 6         |
| <b>2</b> | <b>Installation</b>                                 | <b>7</b>  |
| 2.1      | Hardware Recommendations . . . . .                  | 7         |
| 2.2      | Installing Debian/Ubuntu Packages . . . . .         | 8         |
| 2.3      | Installing RPM Packages . . . . .                   | 9         |
| 2.4      | Installing Chef . . . . .                           | 10        |
| 2.5      | Installing OpenStack . . . . .                      | 14        |
| <b>3</b> | <b>Configuration</b>                                | <b>15</b> |
| 3.1      | Hard Disk and File System Recommendations . . . . . | 15        |
| 3.2      | Ceph Configuration Files . . . . .                  | 16        |
| 3.3      | Deploying with <code>mkcephfs</code> . . . . .      | 39        |
| 3.4      | Deploying with Chef . . . . .                       | 40        |
| 3.5      | Storage Pools . . . . .                             | 44        |
| 3.6      | Authentication . . . . .                            | 44        |
| <b>4</b> | <b>Operating a Cluster</b>                          | <b>47</b> |
| 4.1      | Starting a Cluster . . . . .                        | 48        |
| 4.2      | Checking Cluster Health . . . . .                   | 48        |
| 4.3      | Stopping a Cluster . . . . .                        | 48        |
| <b>5</b> | <b>Ceph FS</b>                                      | <b>49</b> |
| 5.1      | Mount Ceph FS with the Kernel Driver . . . . .      | 49        |
| 5.2      | Mount Ceph FS as a FUSE . . . . .                   | 49        |
| 5.3      | Mount Ceph FS in your File Systems Table . . . . .  | 50        |
| <b>6</b> | <b>Block Devices</b>                                | <b>51</b> |
| 6.1      | RADOS RBD Commands . . . . .                        | 51        |
| 6.2      | RBD Kernel Object Operations . . . . .              | 53        |
| 6.3      | RBD Snapshotting . . . . .                          | 54        |
| 6.4      | QEMU and RBD . . . . .                              | 56        |
| 6.5      | Using <code>libvirt</code> with Ceph RBD . . . . .  | 57        |
| 6.6      | RBD and OpenStack . . . . .                         | 58        |

|           |   |            |
|-----------|---|------------|
| <b>7</b>  | <b>RADOS Gateway</b>                    | <b>59</b>  |
| 7.1       | Install Apache, FastCGI and RADOS GW    | 59         |
| 7.2       | Configuring RADOS Gateway               | 60         |
| 7.3       | RADOS Gateway Configuration Reference   | 62         |
| 7.4       | RADOS S3 API                            | 65         |
| 7.5       | Swift-compatible API                    | 99         |
| <b>8</b>  | <b>Operations</b>                       | <b>117</b> |
| 8.1       | Managing a Ceph cluster                 | 117        |
| 8.2       | Radosgw installation and administration | 130        |
| 8.3       | RBD setup and administration            | 133        |
| 8.4       | Monitoring Ceph                         | 133        |
| <b>9</b>  | <b>Recommendations</b>                  | <b>135</b> |
| 9.1       | Hardware                                | 135        |
| 9.2       | Filesystem                              | 135        |
| 9.3       | Data placement                          | 136        |
| 9.4       | Disabling cryptography                  | 136        |
| <b>10</b> | <b>Control commands</b>                 | <b>137</b> |
| 10.1      | Monitor commands                        | 137        |
| 10.2      | System commands                         | 137        |
| 10.3      | AUTH subsystem                          | 137        |
| 10.4      | PG subsystem                            | 138        |
| 10.5      | OSD subsystem                           | 138        |
| 10.6      | MDS subsystem                           | 141        |
| 10.7      | Mon subsystem                           | 141        |
| <b>11</b> | <b>API Documentation</b>                | <b>143</b> |
| 11.1      | Librados (C)                            | 143        |
| 11.2      | LibradosPP (C++)                        | 171        |
| 11.3      | Librbd (Python)                         | 171        |
| <b>12</b> | <b>Ceph Source Code</b>                 | <b>177</b> |
| 12.1      | Build Prerequisites                     | 177        |
| 12.2      | Downloading a Ceph Release Tarball      | 179        |
| 12.3      | Set Up Git                              | 179        |
| 12.4      | Cloning the Ceph Source Code Repository | 180        |
| 12.5      | Building Ceph                           | 180        |
| 12.6      | Build Ceph Packages                     | 181        |
| 12.7      | Contributing Source Code                | 182        |
| <b>13</b> | <b>Internal developer documentation</b> | <b>183</b> |
| 13.1      | Configuration Management System         | 183        |
| 13.2      | CephContext                             | 185        |
| 13.3      | CephFS delayed deletion                 | 185        |
| 13.4      | Documenting Ceph                        | 185        |
| 13.5      | File striping                           | 187        |
| 13.6      | Filestore filesystem compatibility      | 189        |
| 13.7      | Building Ceph Documentation             | 190        |
| 13.8      | Kernel client troubleshooting (FS)      | 192        |
| 13.9      | Library architecture                    | 192        |
| 13.10     | Debug logs                              | 192        |
| 13.11     | Monitor bootstrap                       | 193        |
| 13.12     | Object Store Architecture Overview      | 196        |

|   |            |
|---|------------|
| 13.13 OSD class path issues . . . . .                           | 197        |
| 13.14 Peering . . . . .   | 197        |
| 13.15 Perf counters . . . . .                                   | 200        |
| 13.16 PG (Placement Group) notes . . . . .                      | 203        |
| 13.17 RBD Layering . . . . .                                    | 205        |
| 13.18 OSD developer documentation . . . . .                     | 209        |
| <b>14 Manual pages</b>  | <b>215</b> |
| 14.1 Section 1, executable programs or shell commands . . . . . | 215        |
| 14.2 Section 8, system administration commands . . . . .        | 217        |
| <b>15 Architecture of Ceph</b>                                  | <b>249</b> |
| 15.1 Monitor cluster . . . . .                                  | 249        |
| 15.2 RADOS . . . . .  | 250        |
| 15.3 Ceph filesystem . . . . .                                  | 250        |
| 15.4 radosgw . . . . .  | 251        |
| 15.5 Rados Block Device (RBD) . . . . .                         | 251        |
| 15.6 Client . . . . .   | 251        |
| 15.7 TODO . . . . .   | 252        |
| <b>16 Frequently Asked Questions</b>                            | <b>253</b> |
| 16.1 Is Ceph Production-Quality? . . . . .                      | 253        |
| 16.2 How can I add a question to this list? . . . . .           | 253        |
| <b>17 Academic papers</b>                                       | <b>255</b> |
| <b>18 Release Notes</b>   | <b>257</b> |
| 18.1 v0.48 “argonaut” . . . . .                                 | 257        |
| <b>19 Appendices</b>  | <b>259</b> |
| 19.1 Differences from POSIX . . . . .                           | 259        |
| <b>Python Module Index</b>                                      | <b>261</b> |



Ceph uniquely delivers **object, block, and file storage in one unified system**. Ceph is highly reliable, easy to manage, and free. The power of Ceph can transform your company's IT infrastructure and your ability to manage vast amounts of data. Ceph delivers extraordinary scalability—thousands of clients accessing petabytes to exabytes of data. Ceph leverages commodity hardware and intelligent daemons to accommodate large numbers of storage hosts, which communicate with each other to replicate data, and redistribute data dynamically. Ceph's cluster of monitors oversees the hosts in the Ceph storage cluster to ensure that the storage hosts are running smoothly.







# GETTING STARTED

Welcome to Ceph! The following sections provide information that will help you get started:

## 1.1 5-minute Quick Start

Thank you for trying Ceph! Petabyte-scale data clusters are quite an undertaking. Before delving deeper into Ceph, we recommend setting up a cluster on a single host to explore some of the functionality.

Ceph **5-Minute Quick Start** is intended for use on one machine with a recent Debian/Ubuntu operating system. The intent is to help you exercise Ceph functionality without the deployment overhead associated with a production-ready storage cluster.

### 1.1.1 Install Debian/Ubuntu

Install a recent release of Debian or Ubuntu (e.g., 12.04 precise).

### 1.1.2 Add Ceph Packages

To get the latest Ceph packages, add a release key to APT, add a source location to your `/etc/apt/sources.list`, update your system and install Ceph.

```
wget -q -O- https://raw.githubusercontent.com/ceph/ceph/master/keys/release.asc | sudo apt-key add -  
echo deb http://ceph.com/debian/ $(lsb_release -sc) main | sudo tee /etc/apt/sources.list.d/ceph.list  
sudo apt-get update && sudo apt-get install ceph
```

### 1.1.3 Add a Configuration File

Modify the contents of the following configuration file such that `localhost` is the actual host name, and the monitor IP address is the actual IP address of the host (i.e., not 127.0.0.1). Then, copy the contents of the modified configuration file and save it to `/etc/ceph/ceph.conf`. This file will configure Ceph to operate a monitor, two OSD daemons and one metadata server on your local machine.

```
[osd]  
    osd journal size = 1000  
    filestore xattr use omap = true  
  
[mon.a]  
    host = localhost
```

```
mon addr = 127.0.0.1:6789

[osd.0]
    host = localhost

[osd.1]
    host = localhost

[mds.a]
    host = localhost
```

## 1.1.4 Deploy the Configuration

To deploy the configuration, create a directory for each daemon as follows:

```
sudo mkdir /var/lib/ceph/osd/ceph-0
sudo mkdir /var/lib/ceph/osd/ceph-1
sudo mkdir /var/lib/ceph/mon/ceph-a
sudo mkdir /var/lib/ceph/mds/ceph-a

cd /etc/ceph
sudo mkcephfs -a -c /etc/ceph/ceph.conf -k ceph.keyring
```

## 1.1.5 Start the Ceph Cluster

Once you have deployed the configuration, start the Ceph cluster.

```
sudo service ceph start
```

Check the health of your Ceph cluster to ensure it is ready.

```
ceph health
```

If your cluster echoes back `HEALTH_OK`, you may begin using your cluster.

## 1.2 RBD Quick Start

To use RADOS block devices, you must have a running Ceph cluster. You may execute this quick start on a separate host if you have the Ceph packages and the `/etc/ceph/ceph.conf` file installed with the appropriate IP address and host name settings modified in the `/etc/ceph/ceph.conf` file.

Create a RADOS Block Device image.

```
rbid create foo --size 4096
```

Load the `rbid` client module.

```
sudo modprobe rbd
```

Map the image to a block device.

```
sudo rbd map foo --pool rbd --name client.admin
```

Use the block device. In the following example, create a file system.

```
sudo mkfs.ext4 -m0 /dev/rbd/rbd/foo
```

Mount the file system.

```
sudo mkdir /mnt/myrbd
sudo mount /dev/rbd/rbd/foo /mnt/myrbd
```

## 1.3 Ceph FS Quick Start

To mount the Ceph FS filesystem, you must have a running Ceph cluster. You may execute this quick start on a separate host if you have the Ceph packages and the `/etc/ceph/ceph.conf` file installed with the appropriate IP address and host name settings modified in the `/etc/ceph/ceph.conf` file.

### 1.3.1 Kernel Driver

Mount Ceph FS as a kernel driver.

```
sudo mkdir /mnt/mycephfs
sudo mount -t ceph {ip-address-of-monitor}:6789:/ /mnt/mycephfs
```

### 1.3.2 Filesystem in User Space (FUSE)

Mount Ceph FS as with FUSE. Replace `{username}` with your username.

```
sudo mkdir /home/{username}/cephfs
sudo ceph-fuse -m {ip-address-of-monitor}:6789 /home/{username}/cephfs
```

## 1.4 Get Involved in the Ceph Community!

These are exciting times in the Ceph community! Get involved!

| Channel            | Description   | Contact Info  |
|--------------------|---|---|
| <b>Blog</b>        | Check the Ceph <a href="#">Blog</a> periodically to keep track of Ceph progress and important announcements.  | <a href="http://ceph.com/community/blog/">http://ceph.com/community/blog/</a>   |
| <b>IRC</b>         | As you delve into Ceph, you may have questions or feedback for the Ceph development team. Ceph developers are often available on the #ceph IRC channel particularly during daytime hours in the US Pacific Standard Time zone.  | <ul style="list-style-type: none"><li>• <b>Domain:</b> <code>irc.oftc.net</code></li><li>• <b>Channel:</b> <code>#ceph</code></li></ul>   |
| <b>Email List</b>  | Keep in touch with developer activity by <a href="#">subscribing</a> to the email list at <a href="mailto:ceph-devel@vger.kernel.org">ceph-devel@vger.kernel.org</a> . You can opt out of the email list at any time by <a href="#">unsubscribing</a> . A simple email is all it takes! If you would like to view the archives, go to <a href="#">Gmane</a> . | <ul style="list-style-type: none"><li>• <a href="#">Subscribe</a></li><li>• <a href="#">Unsubscribe</a></li><li>• <a href="#">Gmane</a></li></ul>   |
| <b>Bug Tracker</b> | You can help keep Ceph production worthy by filing and tracking bugs, and providing feature requests using the <a href="#">Bug Tracker</a> .  | <a href="http://tracker.newdream.net/projects/ceph">http://tracker.newdream.net/projects/ceph</a>   |
| <b>Source Code</b> | If you would like to participate in development, bug fixing, or if you just want the very latest code for Ceph, you can get it at <a href="http://github.com">http://github.com</a> . See Ceph Source Code for details on cloning from github.  | <ul style="list-style-type: none"><li>• <a href="http://github.com:ceph/ceph">http://github.com:ceph/ceph</a></li><li>• <a href="http://ceph.com/download">http://ceph.com/download</a></li></ul> |
| <b>Support</b>     | If you have a very specific problem, an immediate need, or if your deployment requires significant help, consider commercial <a href="#">support</a> .  | <a href="http://inktank.com">http://inktank.com</a>   |

## 1.5 Installing Ceph Manually

Ceph is intended for large-scale deployments, but you may install Ceph on a single host. This guide is intended for Debian/Ubuntu Linux distributions.

1. Install Ceph packages
2. Create a `ceph.conf` file. See Ceph Configuration Files for details.
3. Deploy the Ceph configuration. See Deploy with `mkcephfs` for details.
4. Start a Ceph cluster. See Starting a Cluster for details.
5. Mount Ceph FS. See Ceph FS for details.

# INSTALLATION

Storage clusters are the foundation of the Ceph system. Ceph storage hosts provide object storage. Clients access the Ceph storage cluster directly from an application (using `librados`), over an object storage protocol such as Amazon S3 or OpenStack Swift (using `radosgw`), or with a block device (using `rbd`). To begin using Ceph, you must first set up a storage cluster.

You may deploy Ceph with our `mkcephfs` bootstrap utility for development and test environments. For production environments, we recommend deploying Ceph with the Chef cloud management tool.

If your deployment uses OpenStack, you will also need to install OpenStack.

The following sections provide guidance for installing components used with Ceph:

## 2.1 Hardware Recommendations

Ceph runs on commodity hardware and a Linux operating system over a TCP/IP network. The hardware recommendations for different processes/daemons differ considerably.

- **OSDs:** OSD hosts should have ample data storage in the form of a hard drive or a RAID. Ceph OSDs run the RADOS service, calculate data placement with CRUSH, and maintain their own copy of the cluster map. Therefore, OSDs should have a reasonable amount of processing power.
- **Monitors:** Ceph monitor hosts require enough disk space for the cluster map, but usually do not encounter heavy loads. Monitor hosts do not need to be very powerful.
- **Metadata Servers:** Ceph metadata servers distribute their load. However, metadata servers must be capable of serving their data quickly. Metadata servers should have strong processing capability and plenty of RAM.

---

**Note:** If you are not using the Ceph File System, you do not need a meta data server.

---

### 2.1.1 Minimum Hardware Recommendations

Ceph can run on inexpensive commodity hardware. Small production clusters and development clusters can run successfully with modest hardware.

| Process                                      | Criteria  | Minimum Recommended          |
|--|---|------------------------------|
| RAM<br><del>ceph-osd</del><br>Volume Storage | Processor<br>500 MB per daemon<br>1-disk or RAID per daemon | 64-bit AMD-64/i386 dual-core |
| Network                                      | 2-1GB Ethernet NICs   |                              |
| RAM<br><del>ceph-mon</del><br>Disk Space     | Processor<br>1 GB per daemon<br>10 GB per daemon            | 64-bit AMD-64/i386           |
| Network                                      | 2-1GB Ethernet NICs   |                              |
| RAM<br><del>ceph-mds</del><br>Disk Space     | Processor<br>1 GB minimum per daemon<br>1 MB per daemon     | 64-bit AMD-64/i386 quad-core |
| Network                                      | 2-1GB Ethernet NICs   |                              |

## 2.1.2 Production Cluster Example

Production clusters for petabyte scale data storage may also use commodity hardware, but should have considerably more memory, processing power and data storage to account for heavy traffic loads.

A recent (2012) Ceph cluster project is using two fairly robust hardware configurations for Ceph OSDs, and a lighter configuration for monitors.

| Configuration  | Criteria   | Minimum Recommended          |
|--|--|------------------------------|
| RAM<br><del>Dell PE R510</del><br>Volume Storage<br>Client Network | Processor<br>16 GB<br>8-2TB drives. 1-OS 7-Storage<br>2-1GB Ethernet NICs        | 2 64-bit quad-core Xeon CPUs |
| OSD Network  | 2-1GB Ethernet NICs  |                              |
| NIC Mgmt.  | 2-1GB Ethernet NICs  |                              |
| RAM<br><del>Dell PE R515</del><br>Volume Storage<br>OS Storage     | Processor<br>16 GB<br>12-3TB drives. Storage<br>1-500GB drive. Operating System. | 1 hex-core Opteron CPU       |
| Client Network   | 2-1GB Ethernet NICs  |                              |
| OSD Network  | 2-1GB Ethernet NICs  |                              |
| NIC Mgmt.  | 2-1GB Ethernet NICs  |                              |

## 2.2 Installing Debian/Ubuntu Packages

You may install stable release packages (for stable deployments), development release packages (for the latest features), or development testing packages (for development and QA only). Do not add multiple package sources at the same time.

### 2.2.1 Add Stable Release Packages

We build Debian and Ubuntu packages for each stable release of Ceph. These packages are recommended for anyone deploying Ceph in a production environment.

Packages are cryptographically signed with the `release.asc` key. Add our release key to your system's list of trusted keys to avoid a security warning:

```
wget -q -O- https://raw.githubusercontent.com/ceph/ceph/master/keys/release.asc | sudo apt-key add -
```

Add our package repository to your system's list of APT sources. See [the Debian repository](#) for a complete list of distributions supported.

```
echo deb http://ceph.com/debian/ $(lsb_release -sc) main | sudo tee /etc/apt/sources.list.d/ceph.list
```

## 2.2.2 Add Development Release Packages

Our development process generates a new release of Ceph every 3-4 weeks. These packages are faster-moving than the stable releases, as they get new features integrated quickly, while still undergoing several weeks of QA prior to release.

Packages are cryptographically signed with the `release.asc` key. Add our release key to your system's list of trusted keys to avoid a security warning:

```
wget -q -O- https://raw.githubusercontent.com/ceph/ceph/master/keys/release.asc | sudo apt-key add -
```

Add our package repository to your system's list of APT sources. See [the Debian repository](#) for a complete list of distributions supported.

```
echo deb http://ceph.com/debian-testing/ $(lsb_release -sc) main | sudo tee /etc/apt/sources.list.d/ceph.list
```

## 2.2.3 Add Development Testing Packages

We automatically build Debian and Ubuntu packages for current development branches in the Ceph source code repository. These packages are intended for developers and QA only.

Packages are cryptographically signed with the `autobuild.asc` key. Add our autobuild key to your system's list of trusted keys to avoid a security warning:

```
wget -q -O- https://raw.githubusercontent.com/ceph/ceph/master/keys/autobuild.asc \ | sudo apt-key add -
```

Add our package repository to your system's list of APT sources, but replace `{BRANCH}` with the branch you'd like to use (e.g., `chef-3`, `wip-hack`, `master`, `stable`). We support `oneiric` and `precise` distributions.

```
echo deb http://gitbuilder.ceph.com/ceph-deb-$(lsb_release -sc)-x86_64-basic/ref/{BRANCH} $(lsb_release -sc) main | sudo tee /etc/apt/sources.list.d/ceph.list
```

## 2.2.4 Installing Packages

Once you have added either release or development packages to APT, you should update APT's database and install Ceph:

```
sudo apt-get update && sudo apt-get install ceph
```

## 2.3 Installing RPM Packages

We do not yet build RPM packages for Ceph releases. You can build them yourself from the source tree by running:

```
rpmbuild
```

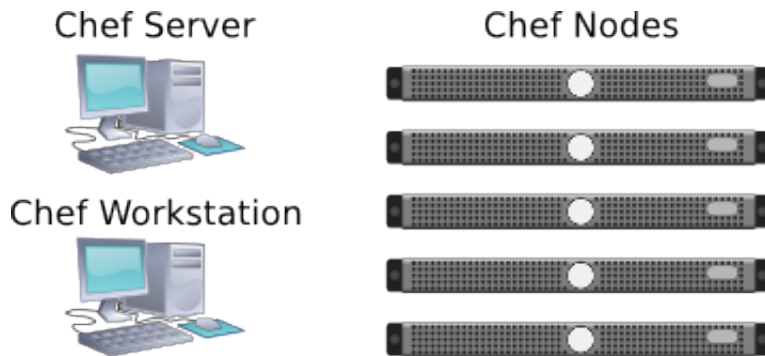
See Ceph Source Code for details. Once you have an RPM, you can install it with:

```
rpm -i ceph-*.rpm
```

## 2.4 Installing Chef

Chef defines three types of entities:

1. **Chef Nodes:** Run `chef-client`, which installs and manages software.
2. **Chef Server:** Interacts with `chef-client` on Chef nodes.
3. **Chef Workstation:** Manages the Chef server.



See [Chef Architecture Introduction](#) for details.

### 2.4.1 Create a chef User

The `chef-client` command requires the proper privileges to install and manage installations. On each Chef node, we recommend creating a `chef` user with full `root` privileges. For example:

```
ssh user@chef-node
sudo useradd -d /home/chef -m chef
sudo passwd chef
```

To provide full privileges, add the following to `/etc/sudoers.d/chef`.

```
echo "chef ALL = (root) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/chef
sudo chmod 0440 /etc/sudoers.d/chef
```

If you are using a version of `sudo` that doesn't support includes, you will need to add the following to the `/etc/sudoers` file:

```
chef ALL = (root) NOPASSWD:ALL
```

---

**Important:** Do not change the file permissions on `/etc/sudoers`. Use a suitable tool such as `visudo`.

---

### 2.4.2 Generate SSH Keys for Chef Clients

Chef's `knife` tool can run `ssh`. To streamline deployments, we recommend generating an SSH key pair without a passphrase for your Chef nodes and copying the public key(s) to your Chef nodes so that you can connect to them from your workstation using `ssh` from `knife` without having to provide a password. To generate a key pair without a passphrase, execute the following on your Chef workstation.



```
ssh-keygen
Generating public/private key pair.
Enter file in which to save the key (/ceph-admin/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /ceph-admin/.ssh/id_rsa.
Your public key has been saved in /ceph-admin/.ssh/id_rsa.pub.
```

You may use RSA or DSA keys. Once you generate your keys, copy them to each OSD host. For example:

```
ssh-copy-id chef@your-node
```

Consider modifying your `~/.ssh/config` file so that it defaults to logging in as `chef` when no username is specified.

```
Host myserver01
    Hostname myserver01.fqdn-or-ip-address.com
    User chef
Host myserver02
    Hostname myserver02.fqdn-or-ip-address.com
    User chef
```

### 2.4.3 Installing Ruby

Chef requires you to install Ruby. Use the version applicable to your current Linux distribution and install Ruby on all of your hosts.

```
sudo apt-get update
sudo apt-get install ruby
```

### 2.4.4 Installing Chef and Chef Server on a Server

If you plan on hosting your [Chef Server at Opscode](#) you may skip this step, but you must make a note of the the fully qualified domain name or IP address of your Chef Server for `knife` and `chef-client`.

First, add Opscode packages to your APT configuration. For example:

```
sudo tee /etc/apt/sources.list.d/chef.list << EOF
deb http://apt.opscode.com/ $(lsb_release -cs)-0.10 main
deb-src http://apt.opscode.com/ $(lsb_release -cs)-0.10 main
EOF
```

Next, you must request keys so that APT can verify the packages. Copy and paste the following line into your command line:

```
sudo touch /etc/apt/trusted.gpg.d/opscode-keyring.gpg && sudo gpg --fetch-key http://apt.opscode.com
```

The key is only used by `apt`, so remove it from the `root` keyring by typing `Y` when prompted to delete it.

Install the Opscode keyring, Chef and Chef server on the host designated as your Chef Server.

```
sudo apt-get update && sudo apt-get upgrade && sudo apt-get install opscode-keyring chef chef-server
```

Enter the fully qualified domain name or IP address for your Chef server. For example:

```
http://fqdn-or-ip-address.com:4000
```

The Chef server installer will prompt you to enter a temporary password. Enter a temporary password (*e.g.*, `foo`) and proceed with the installation.

---

**Tip:** When prompted for a temporary password, you may press **OK**. The installer wants you to re-enter the password to confirm it. To re-enter the password, you must press the **ESC** key.

---

Once the installer finishes and activates the Chef server, you may enter the fully qualified domain name or IP address in a browser to launch the Chef web UI. For example:

```
http://fqdn-or-ip-address.com:4000
```

The Chef web UI will prompt you to enter the username and password.

- **login:** `admin`
- **password:** `foo`

Once you have entered the temporary password, the Chef web UI will prompt you to enter a new password.

## 2.4.5 Install Chef on all Remaining Hosts

Install Chef on all Chef Nodes and on the Chef Workstation (if it is not the same host as the Chef Server). See [Installing Chef Client on Ubuntu or Debian](#) for details.

First, add Opscode packages to your APT configuration. For example:

```
sudo tee /etc/apt/sources.list.d/chef.list << EOF
deb http://apt.opscode.com/ $(lsb_release -cs)-0.10 main
deb-src http://apt.opscode.com/ $(lsb_release -cs)-0.10 main
EOF
```

Next, you must request keys so that APT can verify the packages. Copy and paste the following line into your command line:

```
sudo touch /etc/apt/trusted.gpg.d/opscode-keyring.gpg && sudo gpg --fetch-key http://apt.opscode.com
```

The key is only used by `apt`, so remove it from the `root` keyring by typing `Y` when prompted to delete it.

Install the Opscode keyring and Chef on all hosts other than the Chef Server.

```
sudo apt-get update && sudo apt-get upgrade && sudo apt-get install opscode-keyring chef
```

Enter the fully qualified domain name or IP address for your Chef server. For example:

```
http://fqdn-or-ip-address.com:4000
```

## 2.4.6 Configuring Knife

Once you complete the Chef server installation, install `knife` on the your Chef Workstation. If the Chef server is a remote host, use `ssh` to connect.

```
ssh chef@fqdn-or-ip-address.com
```

In the `/home/chef` directory, create a hidden Chef directory.

```
mkdir -p ~/.chef
```

The server generates validation and web UI certificates with read/write permissions for the user that installed the Chef server. Copy them from the `/etc/chef` directory to the `~/.chef` directory. Then, change their ownership to the current user.

```
sudo cp /etc/chef/validation.pem /etc/chef/webui.pem ~/.chef && sudo chown $(id -u):$(id -g) ~/.chef/
```

From the current user's home directory, configure `knife` with an initial API client.

```
knife configure -i
```

The configuration will prompt you for inputs. Answer accordingly:

*Where should I put the config file? [~/.chef/knife.rb]* Press **Enter** to accept the default value.

*Please enter the chef server URL:* If you are installing the client on the same host as the server, enter `http://localhost:4000`. Otherwise, enter an appropriate URL for the server.

*Please enter a clientname for the new client:* Press **Enter** to accept the default value.

*Please enter the existing admin clientname:* Press **Enter** to accept the default value.

*Please enter the location of the existing admin client's private key:* Override the default value so that it points to the `.chef` directory. (e.g., `/home/chef/.chef/webui.pem`)

*Please enter the validation clientname:* Press **Enter** to accept the default value.

*Please enter the location of the validation key:* Override the default value so that it points to the `.chef` directory. (e.g., `/home/chef/.chef/validation.pem`)

*Please enter the path to a chef repository (or leave blank):* Leave the entry field blank and press **Enter**.

## 2.4.7 Add a Cookbook Path

Add `cookbook_path` to the `~/.chef/knife.rb` configuration file on your Chef workstation. For example:

```
cookbook_path ' /home/{user-name}/chef-cookbooks/ '
```

Then create the path if it doesn't already exist.

```
mkdir /home/{user-name}/chef-cookbooks
```

This is where you will store local copies of cookbooks before uploading them to the Chef server.

## 2.4.8 Copy `validation.pem` to Nodes

Copy the `/etc/chef/validation.pem` file from your Chef server to each Chef Node. In a command line shell on the Chef Server, for each node, replace `{nodename}` in the following line with the node's host name and execute it.

```
sudo cat /etc/chef/validation.pem | ssh {nodename} "exec sudo tee /etc/chef/validation.pem >/dev/null"
```

## 2.4.9 Run `chef-client` on each Chef Node

Run the `chef-client` on each Chef Node so that the nodes register with the Chef server.

```
ssh chef-node
sudo chef-client
```

## 2.4.10 Verify Nodes

Verify that you have setup all the hosts you want to use as Chef nodes.

```
knife node list
```

A list of the nodes you've configured should appear.

See the Deploy With Chef section for information on using Chef to deploy your Ceph cluster.

## 2.5 Installing OpenStack

### 2.5.1 Installing OpenStack with DevStack

To install OpenStack with [DevStack](#), you should ensure that your packages are up to date and properly upgraded.

---

**Tip:** For Ubuntu 12.04 installations, ensure that you upgrade your distribution to the latest release.

---

For example:

```
sudo apt-get update && sudo apt-get upgrade && sudo apt-get dist-upgrade
```

Once you have completed the updates, reboot your system.

Clone the DevStack repository and install OpenStack.

```
git clone git://github.com/openstack-dev/devstack.git
cd devstack; ./stack.sh
```

The installer will prompt you to enter passwords for the various components. Follow the installer to take appropriate notes.

### 2.5.2 Installing OpenStack with Chef

Coming Soon!

### 2.5.3 Installing OpenStack with Crowbar

Coming Soon!

# CONFIGURATION

Ceph can run with a cluster containing thousands of Object Storage Devices (OSDs). A minimal system will have at least two OSDs for data replication. To configure OSD clusters, you must provide settings in the configuration file. Ceph provides default values for many settings, which you can override in the configuration file. Additionally, you can make runtime modification to the configuration using command-line utilities.

When Ceph starts, it activates three daemons:

- `ceph-osd` (mandatory)
- `ceph-mon` (mandatory)
- `ceph-mds` (mandatory for cephfs only)

Each process, daemon or utility loads the host's configuration file. A process may have information about more than one daemon instance (*i.e.*, multiple contexts). A daemon or utility only has information about a single daemon instance (a single context).

---

**Note:** Ceph can run on a single host for evaluation purposes.

---

## 3.1 Hard Disk and File System Recommendations

Ceph aims for data safety, which means that when the application receives notice that data was written to the disk, that data was actually written to the disk. For old kernels (<2.6.33), disable the write cache if the journal is on a raw disk. Newer kernels should work fine.

Use `hdparm` to disable write caching on the hard disk:

```
hdparm -W 0 /dev/hda 0
```

In production environments, we recommend running OSDs with an operating system disk, and a separate disk(s) for data. If you run data and an operating system on a single disk, create a separate partition for your data before configuring your OSD cluster.

Ceph OSDs depend on the Extended Attributes (XATTRs) of the underlying file system for:

- Internal object state
- Snapshot metadata
- RADOS Gateway Access Control Lists (ACLs).

Ceph OSDs rely heavily upon the stability and performance of the underlying file system. The underlying file system must provide sufficient capacity for XATTRs. File system candidates for Ceph include B tree and B+ tree file systems such as:

- `btrfs`
- `XFS`

If you are using `ext4`, mount your file system to enable XATTRs. You must also add the following line to the `[osd]` section of your `ceph.conf` file.

```
filestore xattr use omap = true
```

**Warning:** XATTR limits.

The RADOS Gateway's ACL and Ceph snapshots easily surpass the 4-kilobyte limit for XATTRs in `ext4`, causing the `ceph-osd` process to crash. Version 0.45 or newer uses `leveldb` to bypass this limitation. `ext4` is a poor file system choice if you intend to deploy the RADOS Gateway or use snapshots on versions earlier than 0.45.

---

**Tip:** Use `xfs` initially and `btrfs` when it is ready for production.

The Ceph team believes that the best performance and stability will come from `btrfs`. The `btrfs` file system has internal transactions that keep the local data set in a consistent state. This makes OSDs based on `btrfs` simple to deploy, while providing scalability not currently available from block-based file systems. The 64-kb XATTR limit for `xfs` XATTRs is enough to accommodate RDB snapshot metadata and RADOS Gateway ACLs. So `xfs` is the second-choice file system of the Ceph team in the long run, but `xfs` is currently more stable than `btrfs`. If you only plan to use RADOS and `rbd` without snapshots and without `radosgw`, the `ext4` file system should work just fine.

---

## 3.2 Ceph Configuration Files

When you start the Ceph service, the initialization process activates a series of daemons that run in the background. The hosts in a typical RADOS cluster run at least one of three processes or daemons:

- RADOS (`ceph-osd`)
- Monitor (`ceph-mon`)
- Metadata Server (`ceph-mds`)

Each process or daemon looks for a `ceph.conf` file that provides its configuration settings. The default `ceph.conf` locations in sequential order include:

1. `$CEPH_CONF` (*i.e.*, the path following the `$CEPH_CONF` environment variable)
2. `-c path/path` (*i.e.*, the `-c` command line argument)
3. `/etc/ceph/ceph.conf`
4. `~/.ceph/config`
5. `./ceph.conf` (*i.e.*, in the current working directory)

The `ceph.conf` file provides the settings for each Ceph daemon. Once you have installed the Ceph packages on the OSD Cluster hosts, you need to create a `ceph.conf` file to configure your OSD cluster.

### 3.2.1 Creating `ceph.conf`

The `ceph.conf` file defines:

- Cluster Membership
- Host Names
- Paths to Hosts
- Runtime Options

You can add comments to the `ceph.conf` file by preceding comments with a semi-colon (;). For example:

```
; <--A semi-colon precedes a comment
; A comment may be anything, and always follows a semi-colon on each line.
; We recommend that you provide comments in your configuration file(s).
```

## Configuration File Basics

The `ceph.conf` file configures each instance of the three common processes in a RADOS cluster.

| Setting Scope   | Process  | Setting  | Instance Naming | Description                                   |
|-----------------|----------|----------|-----------------|---|
| All Modules     | All      | [global] | N/A             | Settings affect all instances of all daemons. |
| RADOS           | ceph-osd | [osd]    | Numeric         | Settings affect RADOS instances only.         |
| Monitor         | ceph-mon | [mon]    | Alphanumeric    | Settings affect monitor instances only.       |
| Metadata Server | ceph-mds | [mds]    | Alphanumeric    | Settings affect MDS instances only.           |

## Metavariables

The configuration system supports certain ‘metavariables,’ which are typically used in [global] or process/daemon settings. If metavariables occur inside a configuration value, Ceph expands them into a concrete value—similar to how Bash shell expansion works.

There are a few different metavariables:

| Metavariable | Description   |
|--------------|---|
| \$host       | Expands to the host name of the current daemon.   |
| \$type       | Expands to one of mds, osd, or mon, depending on the type of the current daemon.  |
| \$id         | Expands to the daemon identifier. For <code>osd.0</code> , this would be <code>0</code> ; for <code>mds.a</code> , it would be <code>a</code> . |
| \$num        | Same as <code>\$id</code> .   |
| \$name       | Expands to <code>\$type.\$id</code> .   |
| \$cluster    | Expands to the cluster name. Useful when running multiple clusters on the same hardware.  |

## Global Settings

The Ceph configuration file supports a hierarchy of settings, where child settings inherit the settings of the parent. Global settings affect all instances of all processes in the cluster. Use the [global] setting for values that are common for all hosts in the cluster. You can override each [global] setting by:

1. Changing the setting in a particular process type (e.g., [osd], [mon], [mds] ).
2. Changing the setting in a particular process (e.g., [osd.1] )

Overriding a global setting affects all child processes, except those that you specifically override. For example:

```
[global]
; Enable authentication between hosts within the cluster.
auth supported = cephx
```

## Process/Daemon Settings

You can specify settings that apply to a particular type of process. When you specify settings under `[osd]`, `[mon]` or `[mds]` without specifying a particular instance, the setting will apply to all OSDs, monitors or metadata daemons respectively.

For details on settings for each type of daemon, see the following sections.

### OSD Config Reference

`osd auto upgrade tmap`

**Description** Uses tmap for omap on old objects.

**Type** Boolean

**Default** True

`osd tmapput sets users tmap`

**Description** Uses tmap for debugging only.

**Type** Boolean

**Default** False

`osd data`

**Description**

**Type** String

**Default** None

`osd journal`

**Description**

**Type** String

**Default** None

`osd journal size`

**Description** The size of the journal in MBs.

**Type** 32-bit Int

**Default** 0

`osd max write size`

**Description** The size of the maximum x to write in MBs.

**Type** 32-bit Int

**Default** 90

`osd balance reads`

**Description** Load balance reads?

**Type** Boolean

**Default** False

`osd shed reads`



**Description** Forward from primary to replica.

**Type** 32-bit Int

**Default** False (0)

`osd shed reads min latency`

**Description** The minimum local latency.

**Type** Double

**Default** .01

`osd shed reads min latency diff`

**Description** Percentage difference from peer. 150% default.

**Type** Double

**Default** 1.5

`osd client message size cap`

**Description** Client data allowed in-memory. 500MB default.

**Type** 64-bit Int Unsigned

**Default** 500\*1024L\*1024L

`osd stat refresh interval`

**Description** The status refresh interval in seconds.

**Type** 64-bit Int Unsigned

**Default** .5

`osd pg bits`

**Description** Placement group bits per OSD.

**Type** 32-bit Int

**Default** 6

`osd pgp bits`

**Description** Placement group p bits per OSD?

**Type** 32-bit Int

**Default** 4

`osd pg layout`

**Description** Placement Group bits ? per OSD?

**Type** 32-bit Int

**Default** 2

`osd min rep`

**Description** Need a description.

**Type** 32-bit Int

**Default** 1

`osd max rep`

**Description** Need a description.

**Type** 32-bit Int

**Default** 10

`osd min raid width`

**Description** The minimum RAID width.

**Type** 32-bit Int

**Default** 3

`osd max raid width`

**Description** The maximum RAID width.

**Type** 32-bit Int

**Default** 2

`osd pool default crush rule`

**Description**

**Type** 32-bit Int

**Default** 0

`osd pool default size`

**Description**

**Type** 32-bit Int

**Default** 2

`osd pool default pg num`

**Description**

**Type** 32-bit Int

**Default** 8

`osd pool default pgp num`

**Description**

**Type** 32-bit Int

**Default** 8

`osd map cache max`

**Description**

**Type** 32-bit Int

**Default** 250

`osd map message max`

**Description** max maps per MOSDMap message

**Type** 32-bit Int

**Default** 100

`osd op threads`

**Description** 0 == no threading

**Type** 32-bit Int

**Default** 2

osd disk threads

**Description**

**Type** 32-bit Int

**Default** 1

osd recovery threads

**Description**

**Type** 32-bit Int

**Default** 1

osd recover clone overlap

**Description** preserve clone overlap during rvry/migrat

**Type** Boolean

**Default** false

osd backfill scan min

**Description**

**Type** 32-bit Int

**Default** 64

osd backfill scan max

**Description**

**Type** 32-bit Int

**Default** 512

osd op thread timeout

**Description**

**Type** 32-bit Int

**Default** 30

osd backlog thread timeout

**Description**

**Type** 32-bit Int

**Default** 60\*60\*1

osd recovery thread timeout

**Description**

**Type** 32-bit Int

**Default** 30

osd snap trim thread timeout

**Description****Type** 32-bit Int**Default** 60\*60\*1`osd scrub thread timeout`**Description****Type** 32-bit Int**Default** 60`osd scrub finalize thread timeout`**Description****Type** 32-bit Int**Default** 60\*10`osd remove thread timeout`**Description****Type** 32-bit Int**Default** 60\*60`osd command thread timeout`**Description****Type** 32-bit Int**Default** 10\*60`osd age`**Description****Type** Float**Default** .8`osd age time`**Description****Type** 32-bit Int**Default** 0`osd heartbeat interval`**Description****Type** 32-bit Int**Default** 1`osd mon heartbeat interval`**Description** if no peers | ping monitor**Type** 32-bit Int**Default** 30`osd heartbeat grace`

**Description****Type** 32-bit Int**Default** 20`osd mon report interval max`**Description****Type** 32-bit Int**Default** 120`osd mon report interval min`**Description** pg stats | failures | up thru | boot.**Type** 32-bit Int**Default** 5`osd mon ack timeout`**Description** time out a mon if it doesn't ack stats**Type** 32-bit Int**Default** 30`osd min down reporters`**Description** num OSDs needed to report a down OSD**Type** 32-bit Int**Default** 1`osd min down reports`**Description** num times a down OSD must be reported**Type** 32-bit Int**Default** 3`osd default data pool replay window`**Description****Type** 32-bit Int**Default** 45`osd preserve trimmed log`**Description****Type** Boolean**Default** true`osd auto mark unfound lost`**Description****Type** Boolean**Default** false`osd recovery delay start`

**Description****Type** Float**Default** 15`osd recovery max active`**Description****Type** 32-bit Int**Default** 5`osd recovery max chunk`**Description** max size of push chunk**Type** 64-bit Int Unsigned**Default** 1<<20`osd recovery forget lost objects`**Description** off for now**Type** Boolean**Default** false`osd max scrubs`**Description****Type** 32-bit Int**Default** 1`osd scrub load threshold`**Description****Type** Float**Default** 0.5`osd scrub min interval`**Description****Type** Float**Default** 300`osd scrub max interval`**Description** once a day**Type** Float**Default** 60\*60\*24`osd auto weight`**Description****Type** Boolean**Default** false`osd class error timeout`

**Description** seconds

**Type** Double

**Default** 60.0

osd class timeout

**Description** seconds

**Type** Double

**Default** 60\*60.0

osd class dir

**Description** where rados plugins are stored

**Type** String

**Default** \$libdir/rados-classes

osd check for log corruption

**Description**

**Type** Boolean

**Default** false

osd use stale snap

**Description**

**Type** Boolean

**Default** false

osd rollback to cluster snap

**Description**

**Type** String

**Default** ""

osd default notify timeout

**Description** default notify timeout in seconds

**Type** 32-bit Int Unsigned

**Default** 30

osd kill backfill at

**Description**

**Type** 32-bit Int

**Default** 0

osd min pg log entries

**Description** num entries to keep in pg log when trimming

**Type** 32-bit Int Unsigned

**Default** 1000

osd op complaint time

**Description** how old in secs makes op complaint-worthy

**Type** Float

**Default** 30

osd command max records

**Description**

**Type** 32-bit Int

**Default** 256

## Monitor Config Reference

mon data

**Description**

**Type** String

**Default** ""

mon sync fs threshold

**Description** sync when writing this many objects; 0 to disable.

**Type** 32-bit Integer

**Default** 5

mon tick interval

**Description**

**Type** 32-bit Integer

**Default** 5

mon subscribe interval

**Description**

**Type** Double

**Default** 300

mon osd auto mark in

**Description** mark any booting osds 'in'

**Type** Boolean

**Default** false

mon osd auto mark auto out in

**Description** mark booting auto-marked-out osds 'in'

**Type** Boolean

**Default** true

mon osd auto mark new in

**Description** mark booting new osds 'in'

**Type** Boolean



**Default** true

mon osd down out interval

**Description** seconds

**Type** 32-bit Integer

**Default** 300

mon lease

**Description** lease interval

**Type** Float

**Default** 5

mon lease renew interval

**Description** on leader | to renew the lease

**Type** Float

**Default** 3

mon lease ack timeout

**Description** on leader | if lease isn't acked by all peons

**Type** Float

**Default** 10.0

mon clock drift allowed

**Description** allowed clock drift between monitors

**Type** Float

**Default** .050

mon clock drift warn backoff

**Description** exponential backoff for clock drift warnings

**Type** Float

**Default** 5

mon accept timeout

**Description** on leader | if paxos update isn't accepted

**Type** Float

**Default** 10.0

mon pg create interval

**Description** no more than every 30s

**Type** Float

**Default** 30.0

mon pg stuck threshold

**Description** number of seconds after which pgs can be considered

**Type** 32-bit Integer

**Default** 300

`mon osd full ratio`

**Description** what % full makes an OSD “full”

**Type** Float

**Default** .95

`mon osd nearfull ratio`

**Description** what % full makes an OSD near full

**Type** Float

**Default** .85

`mon globalid prealloc`

**Description** how many globalids to prealloc

**Type** 32-bit Integer

**Default** 100

`mon osd report timeout`

**Description** grace period before declaring unresponsive OSDs dead

**Type** 32-bit Integer

**Default** 900

`mon force standby active`

**Description** should mons force standby-replay mds to be active

**Type** Boolean

**Default** true

`mon min osdmap epochs`

**Description**

**Type** 32-bit Integer

**Default** 500

`mon max pgmap epochs`

**Description**

**Type** 32-bit Integer

**Default** 500

`mon max log epochs`

**Description**

**Type** 32-bit Integer

**Default** 500

`mon probe timeout`

**Description**

**Type** Double

**Default** 2.0

mon slurp timeout

**Description**

**Type** Double

**Default** 10.0

## MDS Config Reference

mds max file size

**Description**

**Type** 64-bit Integer Unsigned

**Default** 1ULL << 40

mds cache size

**Description**

**Type** 32-bit Integer

**Default** 100000

mds cache mid

**Description**

**Type** Float

**Default** 0.7

mds mem max

**Description** // KB

**Type** 32-bit Integer

**Default** 1048576

mds dir commit ratio

**Description**

**Type** Float

**Default** 0.5

mds dir max commit size

**Description** // MB

**Type** 32-bit Integer

**Default** 90

mds decay halflife

**Description**

**Type** Float

**Default** 5

mds beacon interval

**Description****Type** Float**Default** 4

mds beacon grace

**Description****Type** Float**Default** 15

mds blacklist interval

**Description** // how long to blacklist failed nodes**Type** Float**Default** 24.0\*60.0

mds session timeout

**Description** // cap bits and leases time out if client idle**Type** Float**Default** 60

mds session autoclose

**Description** // autoclose idle session**Type** Float**Default** 300

mds reconnect timeout

**Description** // secs to wait for clients during mds restart**Type** Float**Default** 45

mds tick interval

**Description****Type** Float**Default** 5

mds dirstat min interval

**Description** //try to avoid propagating more often than x**Type** Float**Default** 1

mds scatter nudge interval

**Description** // how quickly dirstat changes propagate up**Type** Float**Default** 5

mds client prealloc inos

**Description****Type** 32-bit Integer**Default** 1000

mds early reply

**Description****Type** Boolean**Default** true

mds use tmap

**Description** // use trivialmap for dir updates**Type** Boolean**Default** true

mds default dir hash

**Description** CEPH STR HASH RJENKINS**Type** 32-bit Integer**Default**

mds log

**Description****Type** Boolean**Default** true

mds log skip corrupt events

**Description****Type** Boolean**Default** false

mds log max events

**Description****Type** 32-bit Integer**Default** -1

mds log max segments

**Description** // segment size defined by FileLayout above**Type** 32-bit Integer**Default** 30

mds log max expiring

**Description****Type** 32-bit Integer**Default** 20

mds log eopen size

**Description** // # open inodes per log entry

**Type** 32-bit Integer

**Default** 100

mds bal sample interval

**Description** // every 5 seconds

**Type** Float

**Default** 3

mds bal replicate threshold

**Description**

**Type** Float

**Default** 8000

mds bal unreplicate threshold

**Description**

**Type** Float

**Default** 0

mds bal frag

**Description**

**Type** Boolean

**Default** false

mds bal split size

**Description**

**Type** 32-bit Integer

**Default** 10000

mds bal split rd

**Description**

**Type** Float

**Default** 25000

mds bal split wr

**Description**

**Type** Float

**Default** 10000

mds bal split bits

**Description**

**Type** 32-bit Integer

**Default** 3

mds bal merge size

**Description****Type** 32-bit Integer**Default** 50`mds bal merge rd`**Description****Type** Float**Default** 1000`mds bal merge wr`**Description****Type** Float**Default** 1000`mds bal interval`**Description** // seconds**Type** 32-bit Integer**Default** 10`mds bal fragment interval`**Description** // seconds**Type** 32-bit Integer**Default** 5`mds bal idle threshold`**Description****Type** Float**Default** 0`mds bal max`**Description****Type** 32-bit Integer**Default** -1`mds bal max until`**Description****Type** 32-bit Integer**Default** -1`mds bal mode`**Description****Type** 32-bit Integer**Default** 0`mds bal min rebalance`

**Description** // must be x above avg before we export

**Type** Float

**Default** 0.1

mds bal min start

**Description** // if we need less x. we don't do anything

**Type** Float

**Default** 0.2

mds bal need min

**Description** // take within this range of what we need

**Type** Float

**Default** 0.8

mds bal need max

**Description**

**Type** Float

**Default** 1.2

mds bal midchunk

**Description** // any sub bigger than this taken in full

**Type** Float

**Default** 0.3

mds bal minchunk

**Description** // never take anything smaller than this

**Type** Float

**Default** 0.001

mds bal target removal min

**Description** // min bal iters before old target is removed

**Type** 32-bit Integer

**Default** 5

mds bal target removal max

**Description** // max bal iters before old target is removed

**Type** 32-bit Integer

**Default** 10

mds replay interval

**Description** // time to wait before starting replay again

**Type** Float

**Default** 1

mds shutdown check



**Description****Type** 32-bit Integer**Default** 0

mds thrash exports

**Description****Type** 32-bit Integer**Default** 0

mds thrash fragments

**Description****Type** 32-bit Integer**Default** 0

mds dump cache on map

**Description****Type** Boolean**Default** false

mds dump cache after rejoin

**Description****Type** Boolean**Default** false

mds verify scatter

**Description****Type** Boolean**Default** false

mds debug scatterstat

**Description****Type** Boolean**Default** false

mds debug frag

**Description****Type** Boolean**Default** false

mds debug auth pins

**Description****Type** Boolean**Default** false

mds debug subtrees

**Description****Type** Boolean**Default** false

mds kill mdstable at

**Description****Type** 32-bit Integer**Default** 0

mds kill export at

**Description****Type** 32-bit Integer**Default** 0

mds kill import at

**Description****Type** 32-bit Integer**Default** 0

mds kill link at

**Description****Type** 32-bit Integer**Default** 0

mds kill rename at

**Description****Type** 32-bit Integer**Default** 0

mds wipe sessions

**Description****Type** Boolean**Default** 0

mds wipe ino prealloc

**Description****Type** Boolean**Default** 0

mds skip ino

**Description****Type** 32-bit Integer**Default** 0

max mds

**Description****Type** 32-bit Integer**Default** 1

mds standby for name

**Description****Type** String**Default**

mds standby for rank

**Description****Type** 32-bit Integer**Default** -1

mds standby replay

**Description****Type** Boolean**Default** false**Instance Settings**

You may specify settings for particular instances of an daemon. You may specify an instance by entering its type, delimited by a period (.) and by the instance ID. The instance ID for an OSD is always numeric, but it may be alphanumeric for monitors and metadata servers.

```
[osd.1]
    ; settings affect osd.1 only.
[mon.a1]
    ; settings affect mon.a1 only.
[mds.b2]
    ; settings affect mds.b2 only.
```

**host and addr Settings**

The Hardware Recommendations section provides some hardware guidelines for configuring the cluster. It is possible for a single host to run multiple daemons. For example, a single host with multiple disks or RAID's may run one `ceph-osd` for each disk or RAID. Additionally, a host may run both a `ceph-mon` and an `ceph-osd` daemon on the same host. Ideally, you will have a host for a particular type of process. For example, one host may run `ceph-osd` daemons, another host may run a `ceph-mds` daemon, and other hosts may run `ceph-mon` daemons.

Each host has a name identified by the `host` setting, and a network location (i.e., domain name or IP address) identified by the `addr` setting. For example:

```
[mon.a]
    host = hostName
    mon addr = 150.140.130.120:6789
[osd.0]
    host = hostName
```

## Monitor Configuration

Ceph typically deploys with 3 monitors to ensure high availability should a monitor instance crash. An odd number of monitors (3) ensures that the Paxos algorithm can determine which version of the cluster map is the most accurate.

---

**Note:** You may deploy Ceph with a single monitor, but if the instance fails, the lack of a monitor may interrupt data service availability.

---

Ceph monitors typically listen on port 6789. For example:

```
[mon.a]
    host = hostName
    mon addr = 150.140.130.120:6789
```

## Example Configuration File

```
[global]
    auth supported = cephx

[osd]
    osd journal size = 1000
    ; uncomment the following line if you are mounting with ext4
    ; filestore xattr use omap = true

[mon.a]
    host = myserver01
    mon addr = 10.0.0.101:6789

[mon.b]
    host = myserver02
    mon addr = 10.0.0.102:6789

[mon.c]
    host = myserver03
    mon addr = 10.0.0.103:6789

[osd.0]
    host = myserver01

[osd.1]
    host = myserver02

[osd.2]
    host = myserver03

[mds.a]
    host = myserver01
```

## iptables Configuration

Monitors listen on port 6789, while metadata servers and OSDs listen on the first available port beginning at 6800. Ensure that you open port 6789 on hosts that run a monitor daemon, and open one port beginning at port 6800 for each OSD or metadata server that runs on the host. For example:

```
iptables -A INPUT -m multiport -p tcp -s 192.168.1.0/24 --dports 6789,6800:6803 -j ACCEPT
```

## 3.3 Deploying with mkcephfs

### 3.3.1 Enable Login to Cluster Hosts as root

To deploy with `mkcephfs`, you will need to be able to login as `root` on each host without a password. For each host, perform the following:

```
sudo passwd root
```

Enter a password for the root user.

On the admin host, generate an `ssh` key without specifying a passphrase and use the default locations.

```
ssh-keygen
Generating public/private key pair.
Enter file in which to save the key (/ceph-admin/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /ceph-admin/.ssh/id_rsa.
Your public key has been saved in /ceph-admin/.ssh/id_rsa.pub.
```

You may use RSA or DSA keys. Once you generate your keys, copy them to each OSD host. For example:

```
ssh-copy-id root@myserver01
ssh-copy-id root@myserver02
```

Modify your `~/.ssh/config` file to login as `root`, as follows:

```
Host myserver01
    Hostname myserver01.fully-qualified-domain.com
    User root
Host myserver02
    Hostname myserver02.fully-qualified-domain.com
    User root
```

### 3.3.2 Copy Configuration File to All Hosts

Ceph's `mkcephfs` deployment script does not copy the configuration file you created from the Administration host to the OSD Cluster hosts. Copy the configuration file you created (*i.e.*, `mycluster.conf` in the example below) from the Administration host to `etc/ceph/ceph.conf` on each OSD Cluster host if you are using `mkcephfs` to deploy Ceph.

```
cd /etc/ceph
ssh myserver01 sudo tee /etc/ceph/ceph.conf <ceph.conf
ssh myserver02 sudo tee /etc/ceph/ceph.conf <ceph.conf
ssh myserver03 sudo tee /etc/ceph/ceph.conf <ceph.conf
```

### 3.3.3 Create the Default Directories

The `mkcephfs` deployment script does not create the default server directories. Create server directories for each instance of a Ceph daemon. The `host` variables in the `ceph.conf` file determine which host runs each instance of a Ceph daemon. Using the exemplary `ceph.conf` file, you would perform the following:

On myserver01:

```
sudo mkdir /var/lib/ceph/osd/ceph-0
sudo mkdir /var/lib/ceph/mon/ceph-a
```

On myserver02:

```
sudo mkdir /var/lib/ceph/osd/ceph-1
sudo mkdir /var/lib/ceph/mon/ceph-b
```

On myserver03:

```
sudo mkdir /var/lib/ceph/osd/ceph-2
sudo mkdir /var/lib/ceph/mon/ceph-c
sudo mkdir /var/lib/ceph/mds/ceph-a
```

### 3.3.4 Run mkcephfs

Once you have copied your Ceph Configuration to the OSD Cluster hosts and created the default directories, you may deploy Ceph with the `mkcephfs` script.

---

**Note:** `mkcephfs` is a quick bootstrapping tool. It does not handle more complex operations, such as upgrades.

---

For production environments, deploy Ceph using Chef cookbooks. To run `mkcephfs`, execute the following:

```
cd /etc/ceph
sudo mkcephfs -a -c /etc/ceph/ceph.conf -k ceph.keyring
```

The script adds an admin key to the `ceph.keyring`, which is analogous to a root password. See [Authentication](#) when running with `cephx` enabled.

When you start or stop your cluster, you will not have to use `sudo` or provide passwords. For example:

```
service ceph -a start
```

See [Start | Stop the Cluster](#) for details.

## 3.4 Deploying with Chef

We use Chef cookbooks to deploy Ceph. See [Managing Cookbooks with Knife](#) for details on using `knife`. For Chef installation instructions, see [Installing Chef](#).

### 3.4.1 Clone the Required Cookbooks

To get the cookbooks for Ceph, clone them from git.:

```
cd ~/chef-cookbooks
git clone https://github.com/opscode-cookbooks/apache2.git
git clone https://github.com/ceph/ceph-cookbooks.git ceph
```

### 3.4.2 Add the Required Cookbook Paths

If you added a default cookbook path when you installed Chef, `knife` may be able to upload the cookbook you've cloned to your cookbook path directory without further configuration. If you used a different path, or if the cookbook repository you cloned has a different tree structure, add the required cookbook path to your `knife.rb` file. The `cookbook_path` setting takes a string or an array of strings. For example, you can replace a string path with an array of string paths:

```
cookbook_path '/home/{user-name}/chef-cookbooks/'
```

Becomes:

```
cookbook_path [
  '/home/{user-name}/chef-cookbooks/',
  '/home/{user-name}/chef-cookbooks/{another-directory}',
  '/some/other/path/to/cookbooks/'
]
```

### 3.4.3 Install the Cookbooks

To install Ceph, you must upload the Ceph cookbooks and the Apache cookbooks (for use with RADOSGW) to your Chef server.

```
knife cookbook upload apache2 ceph
```

### 3.4.4 Configure your Ceph Environment

The Chef server can support installation of software for multiple environments. The environment you create for Ceph requires an `fsid`, the secret for your monitor(s) if you are running Ceph with `cephx` authentication, and the host name (i.e., short name) for your monitor hosts.

For the filesystem ID, use `uuidgen` from the `uuid-runtime` package to generate a unique identifier.

```
uuidgen -r
```

For the monitor(s) secret(s), use `ceph-authtool` to generate the secret(s):

```
sudo apt-get update
sudo apt-get install ceph-common
ceph-authtool /dev/stdout --name=mon. --gen-key
```

The secret is the value to the right of "`key =`", and should look something like this:

```
AQBAMuJPINJgFhAAziXIrLvTvAz4PRo5IK/Log==
```

To create an environment for Ceph, set a command line editor. For example:

```
export EDITOR=vim
```

Then, use `knife` to create an environment.

```
knife environment create {env-name}
```

For example:

```
knife environment create Ceph
```

A JSON file will appear. Perform the following steps:

1. Enter a description for the environment.
2. In `"default_attributes": {}`, add `"ceph" : {}`.
3. Within `"ceph" : {}`, add `"monitor-secret":`.
4. Immediately following `"monitor-secret":` add the key you generated within quotes, followed by a comma.
5. Within `"ceph": {}` and following the `monitor-secret` key-value pair, add `"config": {}`
6. Within `"config": {}` add `"fsid":`.
7. Immediately following `"fsid":`, add the unique identifier you generated within quotes, followed by a comma.
8. Within `"config": {}` and following the `fsid` key-value pair, add `"mon_initial_members":`
9. Immediately following `"mon_initial_members":`, enter the initial monitor host names.

For example:

```
"default_attributes" : {
  "ceph": {
    "monitor-secret": "{replace-with-generated-secret}",
    "config": {
      "fsid": "{replace-with-generated-uuid}",
      "mon_initial_members": "{replace-with-monitor-hostname(s) }"
    }
  }
}
```

Advanced users (i.e., developers and QA) may also add `"ceph_branch": "{branch}"` to `default-attributes`, replacing `{branch}` with the name of the branch you wish to use (e.g., `master`).

### 3.4.5 Configure the Roles

Navigate to the Ceph cookbooks directory.

```
cd ~/chef-cookbooks/ceph
```

Create roles for OSDs, monitors, metadata servers, and RADOS Gateways from their respective role files.

```
knife role from file roles/ceph-osd.rb
knife role from file roles/ceph-mon.rb
knife role from file roles/ceph-mds.rb
knife role from file roles/ceph-radosgw.rb
```

### 3.4.6 Configure Nodes

You must configure each node you intend to include in your Ceph cluster. Identify nodes for your Ceph cluster.

```
knife node list
```

For each node you intend to use in your Ceph cluster, configure the node as follows:

```
knife node edit {node-name}
```

The node configuration should appear in your text editor. Change the `chef_environment` value to Ceph (or whatever name you set for your Ceph environment).



In the `run_list`, add `"recipe[ceph::apt]"`, to all nodes as the first setting, so that Chef can install or update the necessary packages. Then, add at least one of:

```
"role[ceph-mon]"
"role[ceph-osd]"
"role[ceph-mds]"
"role[ceph-radosgw]"
```

If you add more than one role, separate them with a comma. Replace the `{hostname}` setting of the `name` key to the host name for the node.

```
{
  "chef_environment": "Ceph",
  "name": "{hostname}",
  "normal": {
    "tags": [

    ]
  },
  "run_list": [
    "recipe[ceph::apt]",
    "role[ceph-mon]",
    "role[ceph-mds]"
  ]
}
```

### 3.4.7 Prepare OSD Disks

For the Ceph 0.48 Argonaut release, install `gdisk` and configure the OSD hard disks for use with Ceph. Replace `{fsid}` with the UUID you generated while using `uuidgen -r`.

```
sudo apt-get install gdisk
sudo sgdisk /dev/{disk} --zap-all --clear --mbrtogpt --largest-new=1 --change-name=1:'ceph data' --ty
```

Create a file system and allocate the disk to your cluster. Specify a filesystem (e.g., `ext4`, `xfs`, `btrfs`). When you execute `ceph-disk-prepare`, remember to replace `{fsid}` with the UUID you generated while using `uuidgen -r`:

```
sudo mkfs -t ext4 /dev/{disk}
sudo mount -o user_xattr /dev/{disk} /mnt
sudo ceph-disk-prepare --cluster-uuid={fsid} /mnt
sudo umount /mnt
```

Finally, simulate a hotplug event.

```
sudo udevadm trigger --subsystem-match=block --action=add
```

### 3.4.8 Run `chef-client` on each Node

Once you have completed the preceding steps, you must run `chef-client` on each node. For example:

```
sudo chef-client
```

### 3.4.9 Proceed to Operating the Cluster

Once you complete the deployment, you may begin operating your cluster. See [Operating a Cluster](#) for details.

## 3.5 Storage Pools

Ceph stores data in ‘pools’ within the OSDs. When you first deploy a cluster without specifying pools, Ceph uses the default pools for storing data. To organize data into pools, see the [rados](#) command for details.

You can list, create, and remove pools. You can also view the pool utilization statistics.

### 3.5.1 List Pools

To list your cluster’s pools, execute:

```
rados lspools
```

The default pools include:

- data
- metadata
- rbd

### 3.5.2 Create a Pool

To create a pool, execute:

```
rados mkpool {pool_name}
```

### 3.5.3 Remove a Pool

To remove a pool, execute:

```
rados rmpool {pool_name}
```

### 3.5.4 Show Pool Stats

To show a pool’s utilization statistics, execute:

```
rados df
```

## 3.6 Authentication

Default users and pools are suitable for initial testing purposes. For test bed and production environments, you should create users and assign pool access to the users.

### 3.6.1 Enabling Authentication

In the `[global]` settings of your `ceph.conf` file, you must enable authentication for your cluster.

```
[global]
    auth supported = cephx
```

The valid values are `cephx` or `none`. If you specify `cephx`, Ceph will look for the keyring in the default search path, which includes `/etc/ceph/keyring`. You can override this location by adding a `keyring` option in the `[global]` section of your `ceph.conf` file, but this is not recommended.

### 3.6.2 The `client.admin` Key

By default, each Ceph command you execute on the command line assumes that you are the `client.admin` default user. When running Ceph with `cephx` enabled, you need to have a `client.admin` key to run `ceph` commands.

The following command will generate and register a `client.admin` key on the monitor with admin capabilities and write it to a keyring on the local file system. If the key already exists, its current value will be returned.

```
sudo ceph auth get-or-create client.admin mds 'allow' osd 'allow *' mon 'allow *' > /etc/ceph/keyring
```

### 3.6.3 Generate a Key

Keys enable a specific user to access the monitor, metadata server and cluster according to capabilities assigned to the key. Capabilities are simple strings specifying some access permissions for a given server type. Each server type has its own string. All capabilities are simply listed in `{type}` and `{capability}` pairs on the command line:

```
sudo ceph auth get-or-create client.{username} {daemon1} {cap1} {daemon2} {cap2} ...
```

For example, to create a user `client.foo` with access `'rw'` for daemon type `'osd'` and `'r'` for daemon type `'mon'`:

```
sudo ceph auth get-or-create client.foo osd rw mon r > keyring.foo
```

### 3.6.4 List Keys in your Cluster

To list the keys registered in your cluster:

```
sudo ceph auth list
```

### 3.6.5 Daemon keyrings

With the exception of the monitors, daemon keyrings are generated in the same way that user keyrings are. By default, the daemons store their keyrings inside their data directory. The default keyring locations, and the capabilities necessary for the daemon to function, are shown below.

| Daemon   | Default keyring location        | Default caps                                |
|----------|---------------------------------|---|
| ceph-mon | <code>\$mon_data/keyring</code> | n/a   |
| ceph-osd | <code>\$osd_data/keyring</code> | mon 'allow rwx' osd 'allow *'               |
| ceph-mds | <code>\$mds_data/keyring</code> | mds 'allow rwx' mds 'allow *' osd 'allow *' |
| radosgw  | <code>\$rgw_data/keyring</code> | mon 'allow r' osd 'allow rwx'               |

Note that the monitor keyring contains a key but no capabilities, and is not part of the cluster auth database.

The daemon data directory locations default to directories of the form:

```
/var/lib/ceph/$daemontype/$cluster-$id
```

For example, `osd.12` would be:

```
/var/lib/ceph/osd/ceph-12
```

You can override these locations, but it is not recommended.

The monitor key can be created with `ceph-authtool` command, and must be identical across all monitors:

```
sudo ceph-authtool {keyring} --create-keyring --gen-key -n mon.
```

# OPERATING A CLUSTER

The `ceph` process provides functionality to **start**, **restart**, and **stop** your Ceph cluster. Each time you execute `ceph`, you must specify at least one option and one command. You may also specify a daemon type or a daemon instance. For most newer Debian/Ubuntu distributions, you may use the following syntax:

```
sudo service ceph [options] [commands] [daemons]
```

For older distributions, you may wish to use the `/etc/init.d/ceph` path:

```
sudo /etc/init.d/ceph [options] [commands] [daemons]
```

The `ceph` options include:

| Option                   | Shortcut        | Description  |
|--------------------------|-----------------|--|
| <code>--verbose</code>   | <code>-v</code> | Use verbose logging.   |
| <code>--valgrind</code>  | N/A             | (Developers only) Use <a href="#">Valgrind</a> debugging.  |
| <code>--allhosts</code>  | <code>-a</code> | Execute on all hosts in <code>ceph.conf</code> . Otherwise, it only executes on <code>localhost</code> . |
| <code>--restart</code>   | N/A             | Automatically restart daemon if it core dumps.   |
| <code>--norestart</code> | N/A             | Don't restart a daemon if it core dumps.   |
| <code>--conf</code>      | <code>-c</code> | Use an alternate configuration file.   |

The `ceph` commands include:

| Command                   | Description   |
|---------------------------|---|
| <code>start</code>        | Start the daemon(s).                                      |
| <code>stop</code>         | Stop the daemon(s).                                       |
| <code>forcestop</code>    | Force the daemon(s) to stop. Same as <code>kill -9</code> |
| <code>killall</code>      | Kill all daemons of a particular type.                    |
| <code>cleanlogs</code>    | Cleans out the log directory.                             |
| <code>cleanalllogs</code> | Cleans out <b>everything</b> in the log directory.        |

The `ceph` daemons include the daemon types:

- `mon`
- `osd`
- `mds`

The `ceph` daemons may also specify a specific instance:

```
sudo /etc/init.d/ceph -a start osd.0
```

Where `osd.0` is the first OSD in the cluster.

## 4.1 Starting a Cluster

To start your Ceph cluster, execute the `ceph` with the `start` command. The usage may differ based upon your Linux distribution. For example, for most newer Debian/Ubuntu distributions, you may use the following syntax:

```
sudo service ceph start [options] [start|restart] [daemonType|daemonID]
```

For older distributions, you may wish to use the `/etc/init.d/ceph` path:

```
sudo /etc/init.d/ceph [options] [start|restart] [daemonType|daemonID]
```

The following examples illustrates a typical use case:

```
sudo service ceph -a start
sudo /etc/init.d/ceph -a start
```

Once you execute with `-a`, Ceph should begin operating. You may also specify a particular daemon instance to constrain the command to a single instance. For example:

```
sudo /etc/init.d/ceph start osd.0
```

## 4.2 Checking Cluster Health

When you start the Ceph cluster, it may take some time to reach a healthy state. You can check on the health of your Ceph cluster with the following:

```
ceph health
```

If you specified non-default locations for your configuration or keyring:

```
ceph -c /path/to/conf -k /path/to/keyring health
```

Upon starting the Ceph cluster, you will likely encounter a health warning such as `HEALTH_WARN XXX num pgs stale`. Wait a few moments and check it again. When your cluster is ready, `ceph health` should return a message such as `HEALTH_OK`. At that point, it is okay to begin using the cluster.

## 4.3 Stopping a Cluster

To stop a cluster, execute one of the following:

```
sudo service -a ceph stop
sudo /etc/init.d/ceph -a stop
```

Ceph should shut down the operating processes.

See Operations for more detailed information.

# CEPH FS

The Ceph FS file system is a POSIX-compliant file system that uses a RADOS cluster to store its data. Ceph FS uses the same RADOS object storage device system as RADOS block devices and RADOS object stores such as the RADOS gateway with its S3 and Swift APIs, or native bindings. Using Ceph FS requires at least one metadata server in your `ceph.conf` configuration file.

## 5.1 Mount Ceph FS with the Kernel Driver

To mount the Ceph file system you may use the `mount` command if you know the monitor host IP address(es), or use the `mount.ceph` utility to resolve the monitor host name(s) into IP address(es) for you. For example:

```
sudo mkdir /mnt/mycephfs
sudo mount -t ceph 192.168.0.1:6789:/ /mnt/mycephfs
```

To mount the Ceph file system with `cephx` authentication enabled, you must specify a user name and a secret.

```
sudo mount -t ceph 192.168.0.1:6789:/ /mnt/mycephfs -o name=admin,secret=AQATSKdNGBnwLhAAAnNDKnH65FmVl
```

The foregoing usage leaves the secret in the Bash history. A more secure approach reads the secret from a file. For example:

```
sudo mount -t ceph 192.168.0.1:6789:/ /mnt/mycephfs -o name=admin,secretfile=/etc/ceph/admin.secret
```

See Authentication for details on `cephx`.

To unmount the Ceph file system, you may use the `umount` command. For example:

```
sudo umount /mnt/mycephfs
```

---

**Tip:** Ensure that you are not within the file system directories before executing this command.

---

See `mount.ceph` for details.

## 5.2 Mount Ceph FS as a FUSE

To mount the Ceph file system as a File System in User Space (FUSE), you may use the `ceph-fuse` command. For example:

```
sudo mkdir /home/username/cephfs
sudo ceph-fuse -m 192.168.0.1:6789 /home/username/cephfs
```

If `cephx` authentication is on, `ceph-fuse` will retrieve the name and secret from the key ring automatically. See `ceph-fuse` for details.

## 5.3 Mount Ceph FS in your File Systems Table

If you mount Ceph FS in your file systems table, the Ceph file system will mount automatically on startup. To mount Ceph FS in your file systems table, add the following to `/etc/fstab`:

```
{ipaddress}:{port}:/ {mount}/{mountpoint} {filesystem-name} [name=username,secret=secretkey|secretfile=secretfile]
```

For example:

```
10.10.10.10:6789:/ /mnt/ceph ceph name=admin,secretfile=/etc/ceph/secret.key,noauto,rw,noexec
```

---

**Important:** The `name` and `secret` or `secretfile` options are mandatory when you have Ceph authentication running. See [Authentication](#) for details.

---



---

# BLOCK DEVICES

A block is a sequence of bytes (for example, a 512-byte block of data). Block-based storage interfaces are the most common way to store data with rotating media such as hard disks, CDs, floppy disks, and even traditional 9-track tape. The ubiquity of block device interfaces makes a virtual block device an ideal candidate to interact with a mass data storage system like Ceph.

Ceph's RADOS Block Devices (RBD) interact with RADOS OSDs using the `librados` and `librbd` libraries. RBDs are thin-provisioned, resizable and store data striped over multiple OSDs in a Ceph cluster. RBDs inherit `librados` capabilities such as snapshotting and cloning. Ceph's RBDs deliver high performance with infinite scalability to kernel objects, kernel virtual machines and cloud-based computing systems like OpenStack and CloudStack.

The `librbd` library converts data blocks into objects for storage in RADOS OSD clusters—the same storage system for `librados` object stores and the Ceph FS filesystem. You can use the same cluster to operate object stores, the Ceph FS filesystem, and RADOS block devices simultaneously.

---

**Important:** To use RBD, you must have a running Ceph cluster.

---

## 6.1 RADOS RBD Commands

The `rbid` command enables you to create, list, introspect and remove block device images. You can also use it to clone images, create snapshots, rollback an image to a snapshot, view a snapshot, etc. For details on using the `rbid` command, see RBD – Manage RADOS Block Device (RBD) Images for details.

---

**Important:** To use RBD commands, you must have a running Ceph cluster.

---

### 6.1.1 Creating a Block Device Image

Before you can add a block device to a Ceph client, you must create an image for it in the OSD cluster first. To create a block device image, execute the following:

```
rbid create {image-name} --size {megabytes} --dest-pool {pool-name}
```

For example, to create a 1GB image named `foo` that stores information in a pool named `swimmingpool`, execute the following:

```
rbid create foo --size 1024
rbid create bar --size 1024 --pool swimmingpool
```

---

**Note:** You must create a pool first before you can specify it as a source. See Storage Pools for details.

---

## 6.1.2 Listing Block Device Images

To list block devices in the `rbid` pool, execute the following:

```
rbid ls
```

To list block devices in a particular pool, execute the following, but replace `{poolname}` with the name of the pool:

```
rbid ls {poolname}
```

For example:

```
rbid ls swimmingpool
```

## 6.1.3 Retrieving Image Information

To retrieve information from a particular image, execute the following, but replace `{image-name}` with the name for the image:

```
rbid --image {image-name} info
```

For example:

```
rbid --image foo info
```

To retrieve information from an image within a pool, execute the following, but replace `{image-name}` with the name of the image and replace `{pool-name}` with the name of the pool:

```
rbid --image {image-name} -p {pool-name} info
```

For example:

```
rbid --image bar -p swimmingpool info
```

## 6.1.4 Resizing a Block Device Image

RBD images are thin provisioned. They don't actually use any physical storage until you begin saving data to them. However, they do have a maximum capacity that you set with the `--size` option. If you want to increase (or decrease) the maximum size of a RADOS block device image, execute the following:

```
rbid resize --image foo --size 2048
```

## 6.1.5 Removing a Block Device Image

To remove a block device, execute the following, but replace `{image-name}` with the name of the image you want to remove:

```
rbid rm {image-name}
```

For example:

```
rbd rm foo
```

To remove a block device from a pool, execute the following, but replace `{image-name}` with the name of the image to remove and replace `{pool-name}` with the name of the pool:

```
rbd rm {image-name} -p {pool-name}
```

For example:

```
rbd rm bar -p swimmingpool
```

---

## 6.2 RBD Kernel Object Operations

---

**Important:** To use kernel object operations, you must have a running Ceph cluster.

---

### 6.2.1 Load the Ceph RBD Module

To map an RBD image to a kernel object, first load the Ceph RBD module:

```
modprobe rbd
```

### 6.2.2 Get a List of RBD Images

To mount an RBD image, first return a list of the images.

```
rbd list
```

### 6.2.3 Map a Block Device with `rbd`

Use `rbd` to map an image name to a kernel object. You must specify the image name, the pool name, and the client name. If you use `cephx` authentication, you must also specify a secret.

```
sudo rbd map {image-name} --pool {pool-name} --name {client-name} --secret {client-secret}
```

For example:

```
sudo rbd map foo --pool rbd --name client.admin
```

If you use `cephx` authentication, you must also specify a secret.

```
echo "10.20.30.40 name=admin,secret=/path/to/secret rbd foo" | sudo tee /sys/bus/rbd/add
```

### 6.2.4 Map a Block Device with `add`

To map an RBD image to a kernel object directly, enter the IP address of the monitor, the user name, and the RBD image name as follows:

```
echo "{mon-ip-address} name={user-name} rbd {image-name}" | sudo tee /sys/bus/rbd/add
```

For example:

```
echo "10.20.30.40 name=admin rbd foo" | sudo tee /sys/bus/rbd/add
```

If you use `cephx` authentication, you must also specify a secret.

```
echo "10.20.30.40 name=admin,secret=/path/to/secret rbd foo" | sudo tee /sys/bus/rbd/add
```

A kernel block device resides under the `/sys/bus/rbd/devices` directory and provides the following functions:

| Function                  | Description   |
|---------------------------|---|
| <code>client_id</code>    | Returns the client ID of the given device ID.         |
| <code>create_snap</code>  | Creates a snap from a snap name and a device ID.      |
| <code>current_snap</code> | Returns the most recent snap for the given device ID. |
| <code>major</code>        |   |
| <code>name</code>         | Returns the RBD image name of the device ID.          |
| <code>pool</code>         | Returns the pool source of the device ID.             |
| <code>refresh</code>      | Refreshes the given device with the SDs.              |
| <code>size</code>         | Returns the size of the device.                       |
| <code>uevent</code>       |   |

## 6.2.5 Show Mapped Block Devices

To show RBD images mapped to kernel block devices with the `rbd` command, specify the `showmapped` option.

```
sudo rbd showmapped
```

Images are mounted as devices sequentially starting from 0. To list all devices mapped to kernel objects, execute the following:

```
ls /sys/bus/rbd/devices
```

## 6.2.6 Unmapping a Block Device

To unmap an RBD image with the `rbd` command, specify the `rm` option and the device name (i.e., by convention the same as the RBD image name).

```
sudo rbd unmap /dev/rbd/{poolname}/{imagename}
```

For example:

```
sudo rbd unmap /dev/rbd/rbd/foo
```

To unmap an RBD image from a kernel object, specify its index and use `tee` to call `remove` as follows, but replace `{device-number}` with the number of the device you want to remove:

```
echo {device-number} | sudo tee /sys/bus/rbd/remove
```

## 6.3 RBD Snapshotting

One of the advanced features of RADOS block devices is that you can create snapshots of the images to retain a history of an image's state. Ceph supports RBD snapshots from the `rbd` command, from a kernel object, from a KVM, and from cloud solutions. Once you create snapshots of an image, you can rollback to a snapshot, list snapshots, remove snapshots and purge the snapshots.

---

**Important:** To use RBD snapshots, you must have a running Ceph cluster.

---

---

**Important:** Generally, you should stop i/o before snapshotting an image. If the image contains a filesystem, the filesystem should be in a consistent state before snapshotting too.

---

### 6.3.1 Create Snapshot

To create a snapshot with `rbd`, specify the `snap create` option, the pool name, the image name and the username. If you use `cephx` for authentication, you must also specify a key or a secret file.

```
rbd --name {user-name} --keyfile=/path/to/secret --pool {pool-name} snap create --snap {snap-name} {image-name}
```

For example:

```
rbd --name client.admin --pool rbd snap create --snap foo.snapname foo
```

### 6.3.2 List Snapshots

To list snapshots of an image, specify the pool name, the image name, and the username. If you use `cephx` for authentication, you must also specify a key or a secret file.

```
rbd --name {user-name} --keyfile=/path/to/secret --pool {pool-name} snap ls {image-name}
```

For example:

```
rbd --name client.admin --pool rbd snap ls foo
```

### 6.3.3 Rollback Snapshot

To rollback a snapshot with `rbd`, specify the `snap rollback` option, the pool name, the image name and the username. If you use `cephx` for authentication, you must also specify a key or a secret file.

```
rbd --name {user-name} --keyfile=/path/to/secret --pool {pool-name} snap rollback --snap {snap-name} {image-name}
```

For example:

```
rbd --name client.admin --pool rbd snap rollback --snap foo.snapname foo
```

### 6.3.4 Delete a Snapshot

To delete a snapshot with `rbd`, specify the `snap rm` option, the pool name, the image name and the username. If you use `cephx` for authentication, you must also specify a key or a secret file.

```
rbd --name {user-name} --keyfile=/path/to/secret --pool {pool-name} snap rm --snap {snap-name} {image-name}
```

For example:

```
rbd --name client.admin --pool rbd snap rm --snap foo.snapname foo
```

## 6.4 QEMU and RBD

Ceph integrates with the QEMU virtual machine. For details on QEMU, see [QEMU Open Source Processor Emulator](#). For QEMU documentation, see [QEMU Manual](#).

---

**Important:** To use RBD with QEMU, you must have a running Ceph cluster.

---

### 6.4.1 Installing QEMU on Ubuntu 12.04 Precise

QEMU packages are incorporated into the Ubuntu 12.04 precise distribution. To install QEMU on precise, execute the following:

```
sudo apt-get install qemu
```

### 6.4.2 Installing QEMU on Earlier Versions of Ubuntu

For Ubuntu distributions 11.10 oneiric and earlier, you must install the 0.15 version of QEMU or later. To build QEMU from source, use the following procedure:

```
cd {your-development-directory}
git clone git://git.qemu.org/qemu.git
cd qemu
./configure --enable-rbd
make; make install
```

### 6.4.3 Creating RBD Images with QEMU

You can create an RBD image from QEMU. You must specify `rbd`, the pool name, and the name of the image you wish to create. You must also specify the size of the image.

```
qemu-img create -f rbd rbd:{pool-name}/{image-name} {size}
```

For example:

```
qemu-img create -f rbd rbd:data/foo 10G
```

### 6.4.4 Resizing RBD Images with QEMU

You can resize an RBD image from QEMU. You must specify `rbd`, the pool name, and the name of the image you wish to resize. You must also specify the size of the image.

```
qemu-img resize -f rbd rbd:{pool-name}/{image-name} {size}
```

For example:

```
qemu-img resize -f rbd rbd:data/foo 10G
```

## 6.4.5 Retrieving RBD Image Information with QEMU

You can retrieve RBD image information from QEMU. You must specify `rbid`, the pool name, and the name of the image.

```
qemu-img info -f rbd rbd:{pool-name}/{image-name}
```

For example:

```
qemu-img info -f rbd rbd:data/foo
```

## 6.5 Using libvirt with Ceph RBD

The `libvirt` library creates a virtual machine abstraction layer between hypervisor interfaces and the software applications that use them. With `libvirt`, developers and system administrators can focus on a common management framework, common API, and common shell interface (i.e., `virsh`) to many different hypervisors, including:

- QEMU/KVM
- XEN
- LXC
- VirtualBox
- etc.

Ceph RADOS block devices support QEMU/KVM, which means you can use RADOS block devices with software that interfaces with `libvirt`. For example, OpenStack's integration to Ceph uses `libvirt` to interact with QEMU/KVM, and QEMU/KVM interacts with RADOS block devices via `librbd`.

See [libvirt Virtualization API](#) for details.

### 6.5.1 Installing libvirt on Ubuntu 12.04 Precise

`libvirt` packages are incorporated into the Ubuntu 12.04 precise distribution. To install `libvirt` on precise, execute the following:

```
sudo apt-get update && sudo apt-get install libvirt-bin
```

### 6.5.2 Installing libvirt on Earlier Versions of Ubuntu

For Ubuntu distributions 11.10 oneiric and earlier, you must build `libvirt` from source. Clone the `libvirt` repository, and use [AutoGen](#) to generate the build. Then execute `make` and `make install` to complete the installation. For example:

```
git clone git://libvirt.org/libvirt.git
cd libvirt
./autogen.sh
make
sudo make install
```

See [libvirt Installation](#) for details.

## 6.6 RBD and OpenStack

You may utilize RBD with OpenStack. To use RBD with OpenStack, you must install QEMU, libvirt, and OpenStack first. We recommend using a separate physical host for your OpenStack installation. OpenStack recommends a minimum of 8GB of RAM and a quad-core processor. If you have not already installed OpenStack, install it now. See [Installing OpenStack](#) for details.

---

**Important:** To use RBD with OpenStack, you must have a running Ceph cluster.

---

### 6.6.1 Create a Pool

By default, RBD uses the `data` pool. You may use any available RBD pool. We recommend creating a pool for Nova. Ensure your Ceph cluster is running, then create a pool.

```
sudo rados mkpool nova
```

### 6.6.2 Install Ceph Common on the OpenStack Host

OpenStack operates as a Ceph client. You must install Ceph common on the OpenStack host, and copy your Ceph cluster's `ceph.conf` file to the `/etc/ceph` directory. If you have installed Ceph on the host, Ceph common is already included.

```
sudo apt-get install ceph-common
cd /etc/ceph
ssh your-openstack-server sudo tee /etc/ceph/ceph.conf <ceph.conf
```

### 6.6.3 Add the RBD Driver and the Pool Name to `nova.conf`

OpenStack requires a driver to interact with RADOS block devices. You must also specify the pool name for the block device. On your OpenStack host, navigate to the `/etc/conf` directory. Open the `nova.conf` file in a text editor using `sudo` privileges and add the following lines to the file:

```
volume_driver=nova.volume.driver.RBDDriver
rbd_pool=nova
```

### 6.6.4 Restart OpenStack

To activate the RBD driver and load the RBD pool name into the configuration, you must restart OpenStack. Navigate the directory where you installed OpenStack, and execute the following:

```
./rejoin-stack.sh
```

If you have OpenStack configured as a service, you can also execute:

```
sudo service nova-volume restart
```

Once OpenStack is up and running, you should be able to create a volume with OpenStack on a Ceph RADOS block device.



# RADOS GATEWAY

RADOS Gateway is an object storage interface built on top of `librados` to provide applications with a RESTful gateway to RADOS clusters. The RADOS Gateway supports two interfaces:

1. **S3-compatible:** Provides block storage functionality with an interface that is compatible with a large subset of the Amazon S3 RESTful API.
2. **Swift-compatible:** Provides block storage functionality with an interface that is compatible with a large subset of the OpenStack Swift API.

RADOS Gateway is a FastCGI module for interacting with `librados`. Since it provides interfaces compatible with OpenStack Swift and Amazon S3, RADOS Gateway has its own user management. RADOS Gateway can store data in the same RADOS cluster used to store data from Ceph FS clients or RADOS block devices. Each interface (S3 or Swift) provides its own namespace.

## 7.1 Install Apache, FastCGI and RADOS GW

To install RADOS Gateway, you must install Apache and FastCGI first.

```
sudo apt-get update && sudo apt-get install apache2 libapache2-mod-fastcgi
```

---

**Note:** The Ceph community provides a slightly optimized version of the `apache2` and `fastcgi` packages. The material difference is that the Ceph packages are optimized for the `100-continue` HTTP response, where the server determines if it will accept the request by first evaluating the request header. See [RFC 2616, Section 8](#) for details on `100-continue`.

---

Enable the URL rewrite modules for Apache and FastCGI. For example:

```
sudo a2enmod rewrite
sudo a2enmod fastcgi
```

By default, the `/etc/apache2/httpd.conf` file is blank. Add a line for the `ServerName` and provide the fully qualified domain name of the host where you will install RADOS GW. For example:

```
ServerName {fqdn}
```

Restart Apache so that the foregoing changes take effect.

```
sudo service apache2 restart
```

Then, install RADOS Gateway. For example:

```
sudo apt-get install radosgw
```

## 7.2 Configuring RADOS Gateway

Before you can start RADOS Gateway, you must modify your `ceph.conf` file to include a section for RADOS Gateway. You must also create an `rgw.conf` file in the `/etc/apache2/sites-enabled` directory. The `rgw.conf` file configures Apache to interact with FastCGI.

### 7.2.1 Add a RADOS GW Configuration to `ceph.conf`

Add the RADOS Gateway configuration to your `ceph.conf` file. The RADOS Gateway configuration requires you to specify the host name where you installed RADOS Gateway, a keyring (for use with `cephx`), the socket path and a log file. For example:

```
[client.radosgw.gateway]
    host = {host-name}
    keyring = /etc/ceph/keyring.rados.gateway
    rgw socket path = /tmp/radosgw.sock
    log file = /var/log/ceph/radosgw.log
```

If you deploy Ceph with `mkcephfs`, manually redeploy `ceph.conf` to the hosts in your cluster. For example:

```
cd /etc/ceph
ssh {host-name} sudo /etc/ceph/ceph.conf < ceph.conf
```

### 7.2.2 Create `rgw.conf`

Create an `rgw.conf` file on the host where you installed RADOS Gateway under the `/etc/apache2/sites-enabled` directory.

We recommend deploying FastCGI as an external server, because allowing Apache to manage FastCGI sometimes introduces high latency. To manage FastCGI as an external server, use the `FastCgiExternalServer` directive. See [FastCgiExternalServer](#) for details on this directive. See [Module mod\\_fastcgi](#) for general details.

```
FastCgiExternalServer /var/www/s3gw.fcgi -socket /tmp/radosgw.sock
```

Once you have configured FastCGI as an external server, you must create the virtual host configuration within your `rgw.conf` file. See [Apache Virtual Host documentation](#) for details on `<VirtualHost>` format and settings. Replace the values in brackets.

```
<VirtualHost *:80>
    ServerName {fqdn}
    ServerAdmin {email.address}
    DocumentRoot /var/www
</VirtualHost>
```

RADOS Gateway requires a rewrite rule for the Amazon S3-compatible interface. It's required for passing in the `HTTP_AUTHORIZATION` env for S3, which is filtered out by Apache. The rewrite rule is not necessary for the OpenStack Swift-compatible interface. Turn on the rewrite engine and add the following rewrite rule to your Virtual Host configuration.

```
RewriteEngine On
RewriteRule ^/([a-zA-Z0-9-_.]*) ([/]?.*) /s3gw.fcgi?page=$1&params=$2&{%QUERY_STRING} [E=HTTP_AUTHORIZA
```

Since the `<VirtualHost>` is running `mod_fastcgi.c`, you must include a section in your `<VirtualHost>` configuration for the `mod_fastcgi.c` module.

```
<VirtualHost *:80>
...
<IfModule mod_fastcgi.c>
    <Directory /var/www>
        Options +ExecCGI
        AllowOverride All
        SetHandler fastcgi-script
        Order allow,deny
        Allow from all
        AuthBasicAuthoritative Off
    </Directory>
</IfModule>
...
</VirtualHost>
```

See [<IfModule> Directive](#) for additional details.

Finally, you should configure Apache to allow encoded slashes, provide paths for log files and to turn off server signatures.

```
<VirtualHost *:80>
...
    AllowEncodedSlashes On
    ErrorLog /var/log/apache2/error.log
    CustomLog /var/log/apache2/access.log combined
    ServerSignature Off
</VirtualHost>
```

## 7.2.3 Enable the RADOS Gateway Configuration

Enable the site for `rgw.conf`.

```
sudo a2ensite rgw.conf
```

Disable the default site.

```
sudo a2dissite default
```

## 7.2.4 Add a RADOS GW Script

Add a `s3gw.fcgi` file (use the same name referenced in the first line of `rgw.conf`) to `/var/www`. The contents of the file should include:

```
#!/bin/sh
exec /usr/bin/radosgw -c /etc/ceph/ceph.conf -n client.rados.gateway
```

Ensure that you apply execute permissions to `s3gw.fcgi`.

```
sudo chmod +x s3gw.fcgi
```

## 7.2.5 Generate a Keyring and Key for RADOS Gateway

You must create a keyring for the RADOS Gateway. For example:

```
sudo ceph-authtool --create-keyring /etc/ceph/keyring.rados.gateway
sudo chmod +r /etc/ceph/keyring.rados.gateway
```

Generate a key so that RADOS Gateway can identify a user name and authenticate the user with the cluster. Then, add capabilities to the key. For example:

```
sudo ceph-authtool /etc/ceph/keyring.rados.gateway -n client.rados.gateway --gen-key
sudo ceph-authtool -n client.rados.gateway --cap osd 'allow rwx' --cap mon 'allow r' /etc/ceph/keyring.rados.gateway
```

## 7.2.6 Add to Ceph Keyring Entries

Once you have created a keyring and key for RADOS GW, add it as an entry in the Ceph keyring. For example:

```
ceph -k /etc/ceph/ceph.keyring auth add client.rados.gateway -i /etc/ceph/keyring.rados.gateway
```

## 7.2.7 Restart Services and Start the RADOS Gateway

To ensure that all components have reloaded their configurations, we recommend restarting your ceph and apaches services. Then, start up the radosgw service. For example:

```
sudo service ceph restart
sudo service apache2 restart
sudo service radosgw start
```

## 7.2.8 Create a RADOS Gateway User

To use the REST interfaces, first create an initial RADOS Gateway user. The RADOS Gateway user is not the same user as the `client.rados.gateway` user, which identifies the RADOS Gateway as a user of the RADOS cluster. The RADOS Gateway user is a user of the RADOS Gateway.

For example:

```
sudo radosgw-admin user create --uid="{username}" --display-name="{Display Name}"
```

For details on RADOS Gateway administration, see `radosgw-admin`.

## 7.3 RADOS Gateway Configuration Reference

The following settings may be added to the `ceph.conf` file. The settings may contain default values. If you do not specify each setting in `ceph.conf`, the default value will be set automatically.

`rgw data`

**Description** Sets the location of the data file for RADOS Gateway.

**Default** `/var/lib/ceph/radosgw/$cluster-$id`

`rgw cache enabled`

**Description** Whether the RADOS Gateway cache is enabled.

**Default** `true`

`rgw cache lru size`

**Description** The number of entries in the RADOS Gateway cache.

**Default** 10000

rgw socket path

**Description** The socket path for the domain socket. `FastCgiExternalServer` uses this socket. If you do not specify a socket path, RADOS Gateway will not run as an external server. The path you specify here must be the same as the path specified in the `rgw.conf` file.

**Default** N/A

**Required** True

**Example** `/var/run/ceph/rgw.sock`

rgw dns name

**Description** The name of the DNS host.

**Default** Same as the host's DNS.

rgw swift url

**Description** The URL for the RADOS Gateway Swift API.

**Default** Same as the host setting.

rgw swift url prefix

**Description** The URL prefix for the Swift API.

**Default** `swift`

**Example** `http://swift.fqdn.com`

rgw enforce swift acls

**Description** Enforces the Swift Access Control List (ACL) settings.

**Default** `true`

rgw print continue

**Description** Enable `100-continue` if it is operational.

**Default** `true`

rgw remote addr param

**Description** The remote address parameter. For example, a variable for the `X-Forwarded-For` address if a reverse proxy is operational.

**Default** `REMOTE_ADDR`

rgw op thread timeout

**Description** The timeout in milliseconds for open threads.

**Default** 600

rgw op thread suicide timeout

**Description** <placeholder>

**Default** <placeholder>

rgw thread pool size

**Description** The size of the thread pool.

**Default** 100 threads.

rgw maintenance tick interval

**Description** <placeholder>

**Default** 10.0

rgw pools preallocate max

**Description** The maximum number of pool to pre-allocate to RADOS Gateway.

**Default** 100

rgw pools preallocate threshold

**Description** The pool pre-allocation threshold. <placeholder>

**Default** 70

rgw log nonexistent bucket

**Description** Should RADOS GW log a request for a non-existent bucket?

**Default** false

rgw log object name

**Description** The logging format for an object name. // man date to see codes (a subset are supported)

**Default** “%Y-%m-%d-%H-%i-%n”

rgw log object name utc

**Description** <placeholder>

**Default** false

rgw usage max shards

**Description** The maximum number of shards.

**Default** 32

rgw usage max user shards

**Description** The maximum number of shards per user.

**Default** 1

rgw enable ops log

**Description** Enable logging for every RGW operation?

**Default** true

rgw enable usage log

**Description** Log bandwidth usage?

**Default** true

rgw usage log flush threshold

**Description** The threshold to flush pending log data.

**Default** 1024

rgw usage log tick interval

**Description** Flush pending log data every n seconds.

**Default** 30

```
rgw intent log object name
```

**Description** The logging format for <placeholder>. // man date to see codes (a subset are supported)

**Default** “%Y-%m-%d-%i-%n”

```
rgw intent log object name utc
```

**Description** Whether the intent log object name should use Coordinated Universal Time (UTC).

**Default** false

```
rgw init timeout
```

**Description** The timeout threshold in seconds.

**Default** 30

```
rgw mime types file
```

**Description** The path and location of the MIME types.

**Default** /etc/mime.types

## 7.4 RADOS S3 API

Ceph supports a RESTful API that is compatible with the the basic data access model of the Amazon S3 API.

### 7.4.1 API

#### Common Entities

##### Bucket and Host Name

There are two different modes of accessing the buckets. The first (preferred) method identifies the bucket as the top-level directory in the URI.

```
GET /mybucket HTTP/1.1
Host: cname.domain.com
```

The second method identifies the bucket via a virtual bucket host name. For example:

```
GET / HTTP/1.1
Host: mybucket.cname.domain.com
```

#### Common Request Headers

| Request Header | Description                     |
|----------------|---------------------------------|
| CONTENT_LENGTH | Length of the request body.     |
| DATE           | Request time and date (in UTC). |
| HOST           | The name of the host server.    |
| AUTHORIZATION  | Authorization token.            |

## Common Response Status

| HTTP Status | Response Code                   |
|-------------|---------------------------------|
| 100         | Continue                        |
| 200         | Success                         |
| 201         | Created                         |
| 202         | Accepted                        |
| 204         | NoContent                       |
| 206         | Partial content                 |
| 304         | NotModified                     |
| 400         | InvalidArgument                 |
| 400         | InvalidDigest                   |
| 400         | BadDigest                       |
| 400         | InvalidBucketName               |
| 400         | InvalidObjectName               |
| 400         | UnresolvableGrantByEmailAddress |
| 400         | InvalidPart                     |
| 400         | InvalidPartOrder                |
| 400         | RequestTimeout                  |
| 400         | EntityTooLarge                  |
| 403         | AccessDenied                    |
| 403         | UserSuspended                   |
| 403         | RequestTimeTooSkewed            |
| 404         | NoSuchKey                       |
| 404         | NoSuchBucket                    |
| 404         | NoSuchUpload                    |
| 405         | MethodNotAllowed                |
| 408         | RequestTimeout                  |
| 409         | BucketAlreadyExists             |
| 409         | BucketNotEmpty                  |
| 411         | MissingContentLength            |
| 412         | PreconditionFailed              |
| 416         | InvalidRange                    |
| 422         | UnprocessableEntity             |
| 500         | InternalError                   |

## Authentication and ACLs

Requests to the RADOS Gateway (RGW) can be either authenticated or unauthenticated. RGW assumes unauthenticated requests are sent by an anonymous user. RGW supports canned ACLs.

### Authentication

Authenticating a request requires including an access key and a Hash-based Message Authentication Code (HMAC) in the request before it is sent to the RGW server. RGW uses an S3-compatible authentication approach.

```
HTTP/1.1
PUT /buckets/bucket/object.mpeg
Host: cname.domain.com
Date: Mon, 2 Jan 2012 00:01:01 +0000
Content-Encoding: mpeg
```



Content-Length: 9999999

Authorization: AWS {access-key}:{hash-of-header-and-secret}

In the foregoing example, replace `{access-key}` with the value for your access key ID followed by a colon (:). Replace `{hash-of-header-and-secret}` with a hash of the header string and the secret corresponding to the access key ID.

To generate the hash of the header string and secret, you must:

1. Get the value of the header string.
2. Normalize the request header string into canonical form.
3. Generate an HMAC using a SHA-1 hashing algorithm. See [RFC 2104](#) and [HMAC](#) for details.
4. Encode the `hmac` result as base-64.

To normalize the header into canonical form:

1. Get all fields beginning with `x-amz-`.
2. Ensure that the fields are all lowercase.
3. Sort the fields lexicographically.
4. Combine multiple instances of the same field name into a single field and separate the field values with a comma.
5. Replace white space and line breaks in field values with a single space.
6. Remove white space before and after colons.
7. Append a new line after each field.
8. Merge the fields back into the header.

Replace the `{hash-of-header-and-secret}` with the base-64 encoded HMAC string.

### Access Control Lists (ACLs)

RGW supports S3-compatible ACL functionality. An ACL is a list of access grants that specify which operations a user can perform on a bucket or on an object. Each grant has a different meaning when applied to a bucket versus applied to an object:

| Permission   | Bucket   | Object                                       |
|--------------|--|--|
| READ         | Grantee can list the objects in the bucket.            | Grantee can read the object.                 |
| WRITE        | Grantee can write or delete objects in the bucket.     | N/A  |
| READ_ACP     | Grantee can read bucket ACL.                           | Grantee can read the object ACL.             |
| WRITE_ACP    | Grantee can write bucket ACL.                          | Grantee can write to the object ACL.         |
| FULL_CONTROL | Grantee has full permissions for object in the bucket. | Grantee can read or write to the object ACL. |

## Service Operations

### List Buckets

`GET /` returns a list of buckets created by the user making the request. `GET /` only returns buckets created by an authenticated user. You cannot make an anonymous request.

**Syntax**

```
GET / HTTP/1.1
```

```
Host: cname.domain.com
```

```
Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

**Response Entities**

| Name                   | Type      | Description  |
|------------------------|-----------|--|
| Buckets                | Container | Container for list of buckets.                         |
| Bucket                 | Container | Container for bucket information.                      |
| Name                   | String    | Bucket name.   |
| CreationDate           | Date      | UTC time when the bucket was created.                  |
| ListAllMyBucketsResult | Container | A container for the result.                            |
| Owner                  | Container | A container for the bucket owner's ID and DisplayName. |
| ID                     | String    | The bucket owner's ID.                                 |
| DisplayName            | String    | The bucket owner's display name.                       |

**Bucket Operations****PUT Bucket**

Creates a new bucket. To create a bucket, you must have a user ID and a valid AWS Access Key ID to authenticate requests. You may not create buckets as an anonymous user.

---

**Note:** We do not support request entities for PUT /{bucket} in this release.

---

**Constraints** In general, bucket names should follow domain name constraints.

- Bucket names must be unique.
- Bucket names must begin and end with a lowercase letter.
- Bucket names may contain a dash (-).

**Syntax**

```
PUT /{bucket} HTTP/1.1
```

```
Host: cname.domain.com
```

```
x-amz-acl: public-read-write
```

```
Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

**Parameters**

| Name      | Description  | Valid Values  | Required |
|-----------|--------------|---|----------|
| x-amz-acl | Canned ACLs. | private, public-read, public-read-write, authenticated-read | No       |

**HTTP Response** If the bucket name is unique, within constraints and unused, the operation will succeed. If a bucket with the same name already exists and the user is the bucket owner, the operation will succeed. If the bucket name is already in use, the operation will fail.

| HTTP Status | Status Code         | Description   |
|-------------|---------------------|---|
| 409         | BucketAlreadyExists | Bucket already exists under different user's ownership. |

## DELETE Bucket

Deletes a bucket. You can reuse bucket names following a successful bucket removal.

### Syntax

```
DELETE /{bucket} HTTP/1.1
Host: cname.domain.com
```

```
Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

| HTTP Response | HTTP Status | Status Code | Description     |
|---------------|-------------|-------------|-----------------|
|               | 204         | No Content  | Bucket removed. |

## GET Bucket

Returns a list of bucket objects.

### Syntax

```
GET /{bucket}?max-keys=25 HTTP/1.1
Host: cname.domain.com
```

| Parameters | Name      | Type    | Description   |
|------------|-----------|---------|---|
|            | prefix    | String  | Only returns objects that contain the specified prefix.           |
|            | delimiter | String  | The delimiter between the prefix and the rest of the object name. |
|            | marker    | String  | A beginning index for the list of objects returned.               |
|            | max-keys  | Integer | The maximum number of keys to return. Default is 1000.            |

| HTTP Response | HTTP Status | Status Code | Description       |
|---------------|-------------|-------------|-------------------|
|               | 200         | OK          | Buckets retrieved |

**Bucket Response Entities** GET /{bucket} returns a container for buckets with the following fields.

| Name             | Type      | Description  |
|------------------|-----------|--|
| ListBucketResult | Entity    | The container for the list of objects.                                       |
| Name             | String    | The name of the bucket whose contents will be returned.                      |
| Prefix           | String    | A prefix for the object keys.  |
| Marker           | String    | A beginning index for the list of objects returned.                          |
| MaxKeys          | Integer   | The maximum number of keys returned.   |
| Delimiter        | String    | If set, objects with the same prefix will appear in the CommonPrefixes list. |
| IsTruncated      | Boolean   | If true, only a subset of the bucket's contents were returned.               |
| CommonPrefixes   | Container | If multiple objects contain the same prefix, they will appear in this list.  |

**Object Response Entities** The `ListBucketResult` contains objects, where each object is within a `Contents` container.

| Name         | Type    | Description                              |
|--------------|---------|--|
| Contents     | Object  | A container for the object.              |
| Key          | String  | The object's key.                        |
| LastModified | Date    | The object's last-modified date/time.    |
| ETag         | String  | An MD-5 hash of the object. (entity tag) |
| Size         | Integer | The object's size.                       |
| StorageClass | String  | Should always return STANDARD.           |

### Get Bucket ACL

Retrieves the bucket access control list. The user needs to be the bucket owner or to have been granted `READ_ACP` permission on the bucket.

**Syntax** Add the `acl` subresource to the bucket request as shown below.

```
GET /{bucket}?acl HTTP/1.1
Host: cname.domain.com
```

```
Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

### Response Entities

| Name                | Type      | Description  |
|---------------------|-----------|--|
| AccessControlPolicy | Container | A container for the response.  |
| AccessControlList   | Container | A container for the ACL information.   |
| Owner               | Container | A container for the bucket owner's ID and <code>DisplayName</code> .                             |
| ID                  | String    | The bucket owner's ID.   |
| DisplayName         | String    | The bucket owner's display name.   |
| Grant               | Container | A container for <code>Grantee</code> and <code>Permission</code> .                               |
| Grantee             | Container | A container for the <code>DisplayName</code> and ID of the user receiving a grant or permission. |
| Permission          | String    | The permission given to the <code>Grantee</code> bucket.   |

### PUT Bucket ACL

Sets an access control to an existing bucket. The user needs to be the bucket owner or to have been granted `WRITE_ACP` permission on the bucket.

**Syntax** Add the `acl` subresource to the bucket request as shown below.

```
PUT /{bucket}?acl HTTP/1.1
```

**Request Entities**

| Name                | Type      | Description   |
|---------------------|-----------|---|
| AccessControlPolicy | Container | A container for the request.  |
| AccessControlList   | Container | A container for the ACL information.  |
| Owner               | Container | A container for the bucket owner's ID and DisplayName.                              |
| ID                  | String    | The bucket owner's ID.  |
| DisplayName         | String    | The bucket owner's display name.  |
| Grant               | Container | A container for Grantee and Permission.   |
| Grantee             | Container | A container for the DisplayName and ID of the user receiving a grant of permission. |
| Permission          | String    | The permission given to the Grantee bucket.   |

**List Bucket Multipart Uploads**

GET `/?uploads` returns a list of the current in-progress multipart uploads—i.e., the application initiates a multipart upload, but the service hasn't completed all the uploads yet.

**Syntax**

GET `/{{bucket}}?uploads` HTTP/1.1

**Parameters** You may specify parameters for GET `/{{bucket}}?uploads`, but none of them are required.

| Name             | Type    | Description   |
|------------------|---------|---|
| prefix           | String  | Returns in-progress uploads whose keys contains the specified prefix.   |
| delimiter        | String  | The delimiter between the prefix and the rest of the object name.   |
| key-marker       | String  | The beginning marker for the list of uploads.   |
| max-keys         | Integer | The maximum number of in-progress uploads. The default is 1000.   |
| max-uploads      | Integer | The maximum number of multipart uploads. The range from 1-1000. The default is 1000.  |
| upload-id-marker | String  | Ignored if <code>key-marker</code> isn't specified. Specifies the ID of first upload to list in lexicographical order at or following the ID. |

## Response Entities

| Name                             | Type      | Description   |
|----------------------------------|-----------|---|
| ListMultipartUploadsResult       | Container | A container for the results.  |
| ListMultipartUploadsResultPrefix | String    | The prefix specified by the prefix request parameter (if any).                        |
| Bucket                           | String    | The bucket that will receive the bucket contents.                                     |
| KeyMarker                        | String    | The key marker specified by the key-marker request parameter (if any).                |
| UploadIdMarker                   | String    | The marker specified by the upload-id-marker request parameter (if any).              |
| NextKeyMarker                    | String    | The key marker to use in a subsequent request if IsTruncated is true.                 |
| NextUploadIdMarker               | String    | The upload ID marker to use in a subsequent request if IsTruncated is true.           |
| MaxUploads                       | Integer   | The max uploads specified by the max-uploads request parameter.                       |
| Delimiter                        | String    | If set, objects with the same prefix will appear in the CommonPrefixes list.          |
| IsTruncated                      | Boolean   | If true, only a subset of the bucket's upload contents were returned.                 |
| Upload                           | Container | A container for Key, UploadId, InitiatorOwner, StorageClass, and Initiated elements.  |
| Key                              | String    | The key of the object once the multipart upload is complete.                          |
| UploadId                         | String    | The ID that identifies the multipart upload.  |
| Initiator                        | Container | Contains the ID and DisplayName of the user who initiated the upload.                 |
| DisplayName                      | String    | The initiator's display name.   |
| ID                               | String    | The initiator's ID.   |
| Owner                            | Container | A container for the ID and DisplayName of the user who owns the uploaded object.      |
| StorageClass                     | String    | The method used to store the resulting object. STANDARD or REDUCED_REDUNDANCY         |
| Initiated                        | Date      | The date and time the user initiated the upload.                                      |
| CommonPrefixes                   | Container | If multiple objects contain the same prefix, they will appear in this list.           |
| CommonPrefixes.Prefix            | String    | The substring of the key after the prefix as defined by the prefix request parameter. |

## Object Operations

## Put Object

Adds an object to a bucket. You must have write permissions on the bucket to perform this operation.

## Syntax

```
PUT /{bucket}/{object} HTTP/1.1
```

| Request Headers | Name                                 | Description                                | Valid Values   | Required |
|-----------------|--------------------------------------|--|--|----------|
|                 | <b>content-md5</b>                   | A base64 encoded MD-5 hash of the message. | A string. No defaults or constraints.  | No       |
|                 | <b>content-type</b>                  | A standard MIME type.                      | Any MIME type. Default: <code>binary/octet-stream</code>   | No       |
|                 | <b>x-amz-meta-<i>&lt;...&gt;</i></b> | User metadata. Stored with the object.     | A string up to 8kb. No defaults.   | No       |
|                 | <b>x-amz-acl</b>                     | A canned ACL.                              | <code>private</code> , <code>public-read</code> , <code>public-read-write</code> , <code>authenticated-read</code> | No       |

## Copy Object

To copy an object, use `PUT` and specify a destination bucket and the object name.

### Syntax

```
PUT /{dest-bucket}/{dest-object} HTTP/1.1
x-amz-copy-source: {source-bucket}/{source-object}
```

| Request Headers | Name                                  | Description                                    | Valid Values   | Required |
|-----------------|---------------------------------------|--|--|----------|
|                 | <b>x-amz-copy-source</b>              | The source bucket name + object name.          | <code>{bucket}/{obj}</code>  | Yes      |
|                 | <b>x-amz-acl</b>                      | A canned ACL.                                  | <code>private</code> , <code>public-read</code> , <code>public-read-write</code> , <code>authenticated-read</code> | No       |
|                 | <b>x-amz-copy-if-modified-since</b>   | Copies only if modified since the timestamp.   | Timestamp  | No       |
|                 | <b>x-amz-copy-if-unmodified-since</b> | Copies only if unmodified since the timestamp. | Timestamp  | No       |
|                 | <b>x-amz-copy-if-match</b>            | Copies only if object ETag matches ETag.       | Entity Tag   | No       |
|                 | <b>x-amz-copy-if-none-match</b>       | Copies only if object ETag doesn't match.      | Entity Tag   | No       |

| Response Entities | Name                    | Type      | Description                                  |
|-------------------|-------------------------|-----------|--|
|                   | <b>CopyObjectResult</b> | Container | A container for the response elements.       |
|                   | <b>LastModified</b>     | Date      | The last modified date of the source object. |
|                   | <b>Etag</b>             | String    | The ETag of the new object.                  |

## Remove Object

Removes an object. Requires `WRITE` permission set on the containing bucket.

### Syntax

```
DELETE /{bucket}/{object} HTTP/1.1
```

## Get Object

Retrieves an object from a bucket within RADOS.

### Syntax

```
GET /{bucket}/{object} HTTP/1.1
```

| Request Headers | Name                       | Description                                    | Valid Values                   | Required |
|-----------------|----------------------------|--|--------------------------------|----------|
|                 | <b>range</b>               | The range of the object to retrieve.           | Range: bytes=beginbyte-endbyte | No       |
|                 | <b>if-modified-since</b>   | Gets only if modified since the timestamp.     | Timestamp                      | No       |
|                 | <b>if-unmodified-since</b> | Gets only if not modified since the timestamp. | Timestamp                      | No       |
|                 | <b>if-match</b>            | Gets only if object ETag matches ETag.         | Entity Tag                     | No       |
|                 | <b>if-none-match</b>       | Gets only if object ETag matches ETag.         | Entity Tag                     | No       |

| Response Headers | Name                 | Description  |
|------------------|----------------------|--|
|                  | <b>Content-Range</b> | Data range, will only be returned if the range header field was specified in the request |

## Get Object Info

Returns information about object. This request will return the same header information as with the Get Object request, but will include the metadata only, not the object data payload.

### Syntax

```
HEAD /{bucket}/{object} HTTP/1.1
```

| Request Headers | Name                       | Description                                    | Valid Values                   | Required |
|-----------------|----------------------------|--|--------------------------------|----------|
|                 | <b>range</b>               | The range of the object to retrieve.           | Range: bytes=beginbyte-endbyte | No       |
|                 | <b>if-modified-since</b>   | Gets only if modified since the timestamp.     | Timestamp                      | No       |
|                 | <b>if-unmodified-since</b> | Gets only if not modified since the timestamp. | Timestamp                      | No       |
|                 | <b>if-match</b>            | Gets only if object ETag matches ETag.         | Entity Tag                     | No       |
|                 | <b>if-none-match</b>       | Gets only if object ETag matches ETag.         | Entity Tag                     | No       |

## Get Object ACL

### Syntax

```
GET /{bucket}/{object}?acl HTTP/1.1
```



**Response Entities**

| Name                | Type      | Description   |
|---------------------|-----------|---|
| AccessControlPolicy | Container | A container for the response.   |
| AccessControlList   | Container | A container for the ACL information.  |
| Owner               | Container | A container for the object owner's ID and DisplayName.                              |
| ID                  | String    | The object owner's ID.  |
| DisplayName         | String    | The object owner's display name.  |
| Grant               | Container | A container for Grantee and Permission.   |
| Grantee             | Container | A container for the DisplayName and ID of the user receiving a grant of permission. |
| Permission          | String    | The permission given to the Grantee object.   |

**Set Object ACL****Syntax**

```
PUT /{bucket}/{object}?acl
```

**Request Entities**

| Name                | Type      | Description   |
|---------------------|-----------|---|
| AccessControlPolicy | Container | A container for the response.   |
| AccessControlList   | Container | A container for the ACL information.  |
| Owner               | Container | A container for the object owner's ID and DisplayName.                              |
| ID                  | String    | The object owner's ID.  |
| DisplayName         | String    | The object owner's display name.  |
| Grant               | Container | A container for Grantee and Permission.   |
| Grantee             | Container | A container for the DisplayName and ID of the user receiving a grant of permission. |
| Permission          | String    | The permission given to the Grantee object.   |

**Initiate Multi-part Upload**

Initiate a multi-part upload process.

**Syntax**

```
POST /{bucket}/{object}?uploads
```

| Request Headers | Name                                       | Description                                | Valid Values   | Required |
|-----------------|--|--|--|----------|
|                 | <b>content-md5</b>                         | A base64 encoded MD-5 hash of the message. | A string. No defaults or constraints.  | No       |
|                 | <b>content-type</b>                        | A standard MIME type.                      | Any MIME type. Default: <code>binary/octet-stream</code>   | No       |
|                 | <b>x-amz-meta-<code>&lt;...&gt;</code></b> | User metadata. Stored with the object.     | A string up to 8kb. No defaults.   | No       |
|                 | <b>x-amz-acl</b>                           | A canned ACL.                              | <code>private</code> , <code>public-read</code> , <code>public-read-write</code> , <code>authenticated-read</code> | No       |

| Response Entities | Name                                     | Type      | Description   |
|-------------------|--|-----------|---|
|                   | InitiatedMultipartUploadConsultContainer | Container | A container for the results.  |
|                   | Bucket                                   | String    | The bucket that will receive the object contents.   |
|                   | Key                                      | String    | The key specified by the <code>key</code> request parameter (if any).                                       |
|                   | UploadId                                 | String    | The ID specified by the <code>upload-id</code> request parameter identifying the multipart upload (if any). |

## Multipart Upload Part

### Syntax

`PUT /{bucket}/{object}?partNumber=&uploadId= HTTP/1.1`

**HTTP Response** The following HTTP response may be returned:

| HTTP Status | Status Code  | Description  |
|-------------|--------------|--|
| <b>404</b>  | NoSuchUpload | Specified upload-id does not match any initiated upload on this object |

## List Multipart Upload Parts

### Syntax

`GET /{bucket}/{object}?uploadId=123 HTTP/1.1`

## Response Entities

| Name                                   | Type      | Description  |
|--|-----------|--|
| InitiatedMultipartUploadCompleteResult | Container | A container for the results.   |
| Bucket                                 | String    | The bucket that will receive the object contents.  |
| Key                                    | String    | The key specified by the key request parameter (if any).   |
| UploadId                               | String    | The ID specified by the upload-id request parameter identifying the multipart upload (if any).   |
| Initiator                              | Container | Contains the ID and DisplayName of the user who initiated the upload.                            |
| ID                                     | String    | The initiator's ID.  |
| DisplayName                            | String    | The initiator's display name.  |
| Owner                                  | Container | A container for the ID and DisplayName of the user who owns the uploaded object.                 |
| StorageClass                           | String    | The method used to store the resulting object. STANDARD or REDUCED_REDUNDANCY                    |
| PartNumberMarker                       | String    | The part marker to use in a subsequent request if IsTruncated is true. Precedes the list.        |
| NextPartNumberMarker                   | String    | The next part marker to use in a subsequent request if IsTruncated is true. The end of the list. |
| MaxParts                               | Integer   | The max parts allowed in the response as specified by the max-parts request parameter.           |
| IsTruncated                            | Boolean   | If true, only a subset of the object's upload contents were returned.                            |
| Part                                   | Container | A container for Key, Part, InitiatorOwner, StorageClass, and Initiated elements.                 |
| PartNumber                             | Integer   | The identification number of the part.   |
| ETag                                   | String    | The part's entity tag.   |
| Size                                   | Integer   | The size of the uploaded part.   |

## Complete Multipart Upload

Assembles uploaded parts and creates a new object, thereby completing a multipart upload.

## Syntax

POST /{bucket}/{object}?uploadId= HTTP/1.1

## Request Entities

| Name                    | Type      | Description                                  | Required |
|-------------------------|-----------|--|----------|
| CompleteMultipartUpload | Container | A container consisting of one or more parts. | Yes      |
| Part                    | Container | A container for the PartNumber and ETag.     | Yes      |
| PartNumber              | Integer   | The identifier of the part.                  | Yes      |
| ETag                    | String    | The part's entity tag.                       | Yes      |

## Response Entities

| Name                          | Type      | Description  |
|-------------------------------|-----------|--|
| CompleteMultipartUploadResult | Container | A container for the response.                        |
| Location                      | URI       | The resource identifier (path) of the new object.    |
| Bucket                        | String    | The name of the bucket that contains the new object. |
| Key                           | String    | The object's key.                                    |
| ETag                          | String    | The entity tag of the new object.                    |

## Abort Multipart Upload

### Syntax

```
DELETE /{bucket}/{object}?uploadId= HTTP/1.1
```

## C++ S3 Examples

### Setup

The following contains includes and globals that will be used in later examples:

```
#include "libs3.h"
#include <stdlib.h>
#include <iostream>
#include <fstream>

const char access_key[] = "ACCESS_KEY";
const char secret_key[] = "SECRET_KEY";
const char host[] = "HOST";
const char sample_bucket[] = "sample_bucket";
const char sample_key[] = "hello.txt";
const char sample_file[] = "resource/hello.txt";

S3BucketContext bucketContext =
{
    host,
    sample_bucket,
    S3ProtocolHTTP,
    S3UriStylePath,
    access_key,
    secret_key
};

S3Status responsePropertiesCallback(
    const S3ResponseProperties *properties,
    void *callbackData)
{
    return S3StatusOK;
}

static void responseCompleteCallback(
    S3Status status,
    const S3ErrorDetails *error,
    void *callbackData)
{
    return;
}

S3ResponseHandler responseHandler =
{
    &responsePropertiesCallback,
    &responseCompleteCallback
};
```

## Creating (and Closing) a Connection

This creates a connection so that you can interact with the server.

```
S3_initialize("s3", S3_INIT_ALL, host);
// Do stuff...
S3_deinitialize();
```

## Listing Owned Buckets

This gets a list of Buckets that you own. This also prints out the bucket name, owner ID, and display name for each bucket.

```
static S3Status listServiceCallback(
    const char *ownerId,
    const char *ownerDisplayName,
    const char *bucketName,
    int64_t creationDate, void *callbackData)
{
    bool *header_printed = (bool*) callbackData;
    if (!*header_printed) {
        *header_printed = true;
        printf("%-22s", "      Bucket");
        printf("  %-20s  %-12s", "      Owner ID", "Display Name");
        printf("\n");
        printf("-----");
        printf("  -----");
        printf("\n");
    }

    printf("%-22s", bucketName);
    printf("  %-20s  %-12s", ownerId ? ownerId : "", ownerDisplayName ? ownerDisplayName : "");
    printf("\n");

    return S3StatusOK;
}

S3ListServiceHandler listServiceHandler =
{
    responseHandler,
    &listServiceCallback
};
bool header_printed = false;
S3_list_service(S3ProtocolHTTP, access_key, secret_key, host, 0, &listServiceHandler, &header_printed);
```

## Creating a Bucket

This creates a new bucket.

```
S3_create_bucket(S3ProtocolHTTP, access_key, secret_key, host, sample_bucket, S3CannedAclPrivate, NULL);
```

## Listing a Bucket's Content

This gets a list of objects in the bucket. This also prints out each object's name, the file size, and last modified date.

```
static S3Status listBucketCallback(
    int isTruncated,
    const char *nextMarker,
    int contentsCount,
    const S3ListBucketContent *contents,
    int commonPrefixesCount,
    const char **commonPrefixes,
    void *callbackData)
{
    printf("%-22s", "      Object Name");
    printf(" %-5s  %-20s", "Size", "  Last Modified");
    printf("\n");
    printf("-----");
    printf("  -----" "  -----");
    printf("\n");

    for (int i = 0; i < contentsCount; i++) {
        char timebuf[256];
        char sizebuf[16];
        const S3ListBucketContent *content = &(contents[i]);
        time_t t = (time_t) content->lastModified;

        strftime(timebuf, sizeof(timebuf), "%Y-%m-%dT%H:%M:%SZ", gmtime(&t));
        sprintf(sizebuf, "%5llu", (unsigned long long) content->size);
        printf("%-22s  %s  %s\n", content->key, sizebuf, timebuf);
    }

    return S3StatusOK;
}

S3ListBucketHandler listBucketHandler =
{
    responseHandler,
    &listBucketCallback
};
S3_list_bucket(&bucketContext, NULL, NULL, NULL, 0, NULL, &listBucketHandler, NULL);
```

The output will look something like this:

```
myphoto1.jpg 251262  2011-08-08T21:35:48.000Z
myphoto2.jpg 262518  2011-08-08T21:38:01.000Z
```

## Deleting a Bucket

---

**Note:** The Bucket must be empty! Otherwise it won't work!

---

```
S3_delete_bucket(S3ProtocolHTTP, S3UriStylePath, access_key, secret_key, host, sample_bucket, NULL, &
```

## Creating an Object (from a file)

This creates a file hello.txt.

```
#include <sys/stat.h>
typedef struct put_object_callback_data
```

```

{
    FILE *infile;
    uint64_t contentLength;
} put_object_callback_data;

static int putObjectDataCallback(int bufferSize, char *buffer, void *callbackData)
{
    put_object_callback_data *data = (put_object_callback_data *) callbackData;

    int ret = 0;

    if (data->contentLength) {
        int toRead = ((data->contentLength > (unsigned) bufferSize) ? (unsigned) bufferSize : data->contentLength);
        ret = fread(buffer, 1, toRead, data->infile);
    }
    data->contentLength -= ret;
    return ret;
}

put_object_callback_data data;
struct stat statbuf;
if (stat(sample_file, &statbuf) == -1) {
    fprintf(stderr, "\nERROR: Failed to stat file %s: ", sample_file);
    perror(0);
    exit(-1);
}

int contentLength = statbuf.st_size;
data.contentLength = contentLength;

if (!(data.infile = fopen(sample_file, "r"))) {
    fprintf(stderr, "\nERROR: Failed to open input file %s: ", sample_file);
    perror(0);
    exit(-1);
}

S3PutObjectHandler putObjectHandler =
{
    responseHandler,
    &putObjectDataCallback
};

S3_put_object(&bucketContext, sample_key, contentLength, NULL, NULL, &putObjectHandler, &data);

```

### Download an Object (to a file)

This downloads a file and prints the contents.

```

static S3Status getObjectDataCallback(int bufferSize, const char *buffer, void *callbackData)
{
    FILE *outfile = (FILE *) callbackData;
    size_t wrote = fwrite(buffer, 1, bufferSize, outfile);
    return ((wrote < (size_t) bufferSize) ? S3StatusAbortedByCallback : S3StatusOK);
}

S3GetObjectHandler getObjectHandler =

```

```
{
    responseHandler,
    &getObjectDataCallback
};
FILE *outfile = stdout;
S3_get_object(&bucketContext, sample_key, NULL, 0, 0, NULL, &getObjectHandler, outfile);
```

## Delete an Object

This deletes an object.

```
S3ResponseHandler deleteResponseHandler =
{
    0,
    &responseCompleteCallback
};
S3_delete_object(&bucketContext, sample_key, 0, &deleteResponseHandler, 0);
```

## Change an Object's ACL

This changes an object's ACL to grant full control to another user.

```
#include <string.h>
char ownerId[] = "owner";
char ownerDisplayName[] = "owner";
char granteeId[] = "grantee";
char granteeDisplayName[] = "grantee";

S3AclGrant grants[] = {
    {
        S3GranteeTypeCanonicalUser,
        {},
        S3PermissionFullControl
    },
    {
        S3GranteeTypeCanonicalUser,
        {},
        S3PermissionReadACP
    },
    {
        S3GranteeTypeAllUsers,
        {},
        S3PermissionRead
    }
};

strncpy(grants[0].grantee.canonicalUser.id, ownerId, S3_MAX_GRANTEE_USER_ID_SIZE);
strncpy(grants[0].grantee.canonicalUser.displayName, ownerDisplayName, S3_MAX_GRANTEE_DISPLAY_NAME_SIZE);

strncpy(grants[1].grantee.canonicalUser.id, granteeId, S3_MAX_GRANTEE_USER_ID_SIZE);
strncpy(grants[1].grantee.canonicalUser.displayName, granteeDisplayName, S3_MAX_GRANTEE_DISPLAY_NAME_SIZE);

S3_set_acl(&bucketContext, sample_key, ownerId, ownerDisplayName, 3, grants, 0, &responseHandler, 0);
```



## Generate Object Download URL (signed)

This generates a signed download URL that will be valid for 5 minutes.

```
#include <time.h>
char buffer[S3_MAX_AUTHENTICATED_QUERY_STRING_SIZE];
int64_t expires = time(NULL) + 60 * 5; // Current time + 5 minutes

S3_generate_authenticated_query_string(buffer, &bucketContext, sample_key, expires, NULL);
```

## C# S3 Examples

### Creating a Connection

This creates a connection so that you can interact with the server.

```
using System;
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;

string accessKey = "put your access key here!";
string secretKey = "put your secret key here!";

AmazonS3Config config = new AmazonS3Config();
config.ServiceURL = "objects.dreamhost.com";

AmazonS3 client = Amazon.AWSClientFactory.CreateAmazonS3Client(
    accessKey,
    secretKey,
    config
);
```

### Listing Owned Buckets

This gets a list of Buckets that you own. This also prints out the bucket name and creation date of each bucket.

```
ListBucketResponse response = client.ListBuckets();
foreach (S3Bucket b in response.Buckets)
{
    Console.WriteLine("{0}\t{1}", b.BucketName, b.CreationDate);
}
```

The output will look something like this:

```
mahbuckat1    2011-04-21T18:05:39.000Z
mahbuckat2    2011-04-21T18:05:48.000Z
mahbuckat3    2011-04-21T18:07:18.000Z
```

### Creating a Bucket

This creates a new bucket called my-new-bucket

```
PutBucketRequest request = new PutBucketRequest();
request.BucketName = "my-new-bucket";
client.PutBucket(request);
```

### Listing a Bucket's Content

This gets a list of objects in the bucket. This also prints out each object's name, the file size, and last modified date.

```
ListObjectsRequest request = new ListObjectsRequest();
request.BucketName = "my-new-bucket";
ListObjectsResponse response = client.ListObjects(request);
foreach (S3Object o in response.S3Objects)
{
    Console.WriteLine("{0}\t{1}\t{2}", o.Key, o.Size, o.LastModified);
}
```

The output will look something like this:

```
myphoto1.jpg 251262 2011-08-08T21:35:48.000Z
myphoto2.jpg 262518 2011-08-08T21:38:01.000Z
```

### Deleting a Bucket

---

**Note:** The Bucket must be empty! Otherwise it won't work!

---

```
DeleteBucketRequest request = new DeleteBucketRequest();
request.BucketName = "my-new-bucket";
client.DeleteBucket(request);
```

### Forced Delete for Non-empty Buckets

**Attention:** not available

### Creating an Object

This creates a file `hello.txt` with the string "Hello World!"

```
PutObjectRequest request = new PutObjectRequest();
request.Bucket = "my-new-bucket";
request.Key = "hello.txt";
request.ContentType = "text/plain";
request.ContentBody = "Hello World!";
client.PutObject(request);
```

### Change an Object's ACL

This makes the object `hello.txt` to be publicly readable, and `secret_plans.txt` to be private.

```
SetACLRequest request = new SetACLRequest();
request.BucketName = "my-new-bucket";
request.Key        = "hello.txt";
request.CannedACL  = S3CannedACL.PublicRead;
client.SetACL(request);

SetACLRequest request2 = new SetACLRequest();
request2.BucketName = "my-new-bucket";
request2.Key        = "secret_plans.txt";
request2.CannedACL  = S3CannedACL.Private;
client.SetACL(request2);
```

### Download an Object (to a file)

This downloads the object `perl_poetry.pdf` and saves it in `C:\Users\larry\Documents`

```
GetObjectRequest request = new GetObjectRequest();
request.BucketName = "my-new-bucket";
request.Key        = "perl_poetry.pdf";
GetObjectResponse response = client.GetObject(request);
response.WriteResponseStreamToFile("C:\\Users\\larry\\Documents\\perl_poetry.pdf");
```

### Delete an Object

This deletes the object `goodbye.txt`

```
DeleteObjectRequest request = new DeleteObjectRequest();
request.BucketName = "my-new-bucket";
request.Key        = "goodbye.txt";
client.DeleteObject(request);
```

### Generate Object Download URLs (signed and unsigned)

This generates an unsigned download URL for `hello.txt`. This works because we made `hello.txt` public by setting the ACL above. This then generates a signed download URL for `secret_plans.txt` that will work for 1 hour. Signed download URLs will work for the time period even if the object is private (when the time period is up, the URL will stop working).

---

**Note:** The C# S3 Library does not have a method for generating unsigned URLs, so the following example only shows generating signed URLs.

---

```
GetPreSignedUrlRequest request = new GetPreSignedUrlRequest();
request.BucketName = "my-bucket-name";
request.Key        = "secret_plans.txt";
request.Expires    = DateTime.Now.AddHours(1);
request.Protocol   = Protocol.HTTP;
string url = client.GetPreSignedURL(request);
Console.WriteLine(url);
```

The output of this will look something like:

`http://objects.dreamhost.com/my-bucket-name/secret_plans.txt?Signature=XXXXXXXXXXXXXXXXXXXXXXXXXXXX&E`

## Java S3 Examples

### Setup

The following examples may require some or all of the following java classes to be imported:

```
import java.io.ByteArrayInputStream;
import java.io.File;
import java.util.List;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.util.StringUtils;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.Bucket;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.S3ObjectSummary;
```

### Creating a Connection

This creates a connection so that you can interact with the server.

```
String accessKey = "insert your access key here!";
String secretKey = "insert your secret key here!";

AWSCredentials credentials = new BasicAWSCredentials(accessKey, secretKey);
AmazonS3 conn = new AmazonS3Client(credentials);
conn.setEndpoint("objects.dreamhost.com");
```

### Listing Owned Buckets

This gets a list of Buckets that you own. This also prints out the bucket name and creation date of each bucket.

```
List<Bucket> buckets = conn.listBuckets();
for (Bucket bucket : buckets) {
    System.out.println(bucket.getName() + "\t" +
        StringUtils.fromDate(bucket.getCreationDate()));
}
```

The output will look something like this:

```
mahbuckat1    2011-04-21T18:05:39.000Z
mahbuckat2    2011-04-21T18:05:48.000Z
mahbuckat3    2011-04-21T18:07:18.000Z
```

## Creating a Bucket

This creates a new bucket called `my-new-bucket`

```
Bucket bucket = conn.createBucket("my-new-bucket");
```

## Listing a Bucket's Content

This gets a list of objects in the bucket. This also prints out each object's name, the file size, and last modified date.

```
ObjectListing objects = conn.listObjects(bucket.getName());
do {
    for (S3ObjectSummary objectSummary : objects.getObjectSummaries()) {
        System.out.println(objectSummary.getKey() + "\t" +
            ObjectSummary.getSize() + "\t" +
            StringUtils.fromDate(objectSummary.getLastModified()));
    }
    objects = conn.listNextBatchOfObjects(objects);
} while (objects.isTruncated());
```

The output will look something like this:

```
myphoto1.jpg 251262 2011-08-08T21:35:48.000Z
myphoto2.jpg 262518 2011-08-08T21:38:01.000Z
```

## Deleting a Bucket

---

**Note:** The Bucket must be empty! Otherwise it won't work!

---

```
conn.deleteBucket(bucket.getName());
```

## Forced Delete for Non-empty Buckets

|                                 |
|---------------------------------|
| <b>Attention:</b> not available |
|---------------------------------|

## Creating an Object

This creates a file `hello.txt` with the string "Hello World!"

```
ByteArrayInputStream input = new ByteArrayInputStream("Hello World!".getBytes());
conn.putObject(bucket.getName(), "hello.txt", input, new ObjectMetadata());
```

## Change an Object's ACL

This makes the object `hello.txt` to be publicly readable, and `secret_plans.txt` to be private.

```
conn.setObjectAcl(bucket.getName(), "hello.txt", CannedAccessControlList.PublicRead);
conn.setObjectAcl(bucket.getName(), "secret_plans.txt", CannedAccessControlList.Private);
```

### Download an Object (to a file)

This downloads the object `perl_poetry.pdf` and saves it in `/home/larry/documents`

```
conn.getObject(  
    new GetObjectRequest(bucket.getName(), "perl_poetry.pdf"),  
    new File("/home/larry/documents/perl_poetry.pdf")  
);
```

### Delete an Object

This deletes the object `goodbye.txt`

```
conn.deleteObject(bucket.getName(), "goodbye.txt");
```

### Generate Object Download URLs (signed and unsigned)

This generates an unsigned download URL for `hello.txt`. This works because we made `hello.txt` public by setting the ACL above. This then generates a signed download URL for `secret_plans.txt` that will work for 1 hour. Signed download URLs will work for the time period even if the object is private (when the time period is up, the URL will stop working).

---

**Note:** The java library does not have a method for generating unsigned URLs, so the example below just generates a signed URL.

---

```
GeneratePresignedUrlRequest request = new GeneratePresignedUrlRequest(bucket.getName(), "secret_plans.txt");  
System.out.println(conn.generatePresignedUrl(request));
```

The output will look something like this:

```
https://my-bucket-name.objects.dreamhost.com/secret_plans.txt?Signature=XXXXXXXXXXXXXXXXXXXXXXXXXXXX&Expires=1456789012
```

## Perl S3 Examples

### Creating a Connection

This creates a connection so that you can interact with the server.

```
use Amazon::S3;  
my $access_key = 'put your access key here!';  
my $secret_key = 'put your secret key here!';  
  
my $conn = Amazon::S3->new({  
    aws_access_key_id => $access_key,  
    aws_secret_access_key => $secret_key,  
    host => 'objects.dreamhost.com',  
    secure => 1,  
    retry => 1,  
});
```

## Listing Owned Buckets

This gets a list of `Amazon::S3::Bucket` objects that you own. We'll also print out the bucket name and creation date of each bucket.

```
my @buckets = @{$conn->buckets->{buckets} || []};
foreach my $bucket (@buckets) {
    print $bucket->bucket . "\t" . $bucket->creation_date . "\n";
}
```

The output will look something like this:

```
mahbucket1    2011-04-21T18:05:39.000Z
mahbucket2    2011-04-21T18:05:48.000Z
mahbucket3    2011-04-21T18:07:18.000Z
```

## Creating a Bucket

This creates a new bucket called `my-new-bucket`

```
my $bucket = $conn->add_bucket({ bucket => 'my-new-bucket' });
```

## Listing a Bucket's Content

This gets a list of hashes with info about each object in the bucket. We'll also print out each object's name, the file size, and last modified date.

```
my @keys = @{$bucket->list_all->{keys} || []};
foreach my $key (@keys) {
    print "$key->{key}\t$key->{size}\t$key->{last_modified}\n";
}
```

The output will look something like this:

```
myphoto1.jpg 251262 2011-08-08T21:35:48.000Z
myphoto2.jpg 262518 2011-08-08T21:38:01.000Z
```

## Deleting a Bucket

---

**Note:** The Bucket must be empty! Otherwise it won't work!

---

```
$conn->delete_bucket($bucket);
```

## Forced Delete for Non-empty Buckets

**Attention:** not available in the `Amazon::S3` perl module

## Creating an Object

This creates a file `hello.txt` with the string "Hello World!"

```
$bucket->add_key(  
    'hello.txt', 'Hello World!',  
    { content_type => 'text/plain' },  
);
```

## Change an Object's ACL

This makes the object `hello.txt` to be publicly readable and `secret_plans.txt` to be private.

```
$bucket->set_acl({  
    key      => 'hello.txt',  
    acl_short => 'public-read',  
});  
$bucket->set_acl({  
    key      => 'secret_plans.txt',  
    acl_short => 'private',  
});
```

## Download an Object (to a file)

This downloads the object `perl_poetry.pdf` and saves it in `/home/larry/documents/`

```
$bucket->get_key_filename('perl_poetry.pdf', undef,  
    '/home/larry/documents/perl_poetry.pdf');
```

## Delete an Object

This deletes the object `goodbye.txt`

```
$bucket->delete_key('goodbye.txt');
```

## Generate Object Download URLs (signed and unsigned)

This generates an unsigned download URL for `hello.txt`. This works because we made `hello.txt` public by setting the ACL above. Then this generates a signed download URL for `secret_plans.txt` that will work for 1 hour. Signed download URLs will work for the time period even if the object is private (when the time period is up, the URL will stop working).

---

**Note:** The `Amazon::S3` module does not have a way to generate download URLs, so we're going to be using another module instead. Unfortunately, most modules for generating these URLs assume that you are using Amazon, so we've had to go with using a more obscure module, `Muck::FS::S3`. This should be the same as Amazon's sample S3 perl module, but this sample module is not in CPAN. So, you can either use CPAN to install `Muck::FS::S3`, or install Amazon's sample S3 module manually. If you go the manual route, you can remove `Muck::FS::` from the example below.

---



```

use Muck::FS::S3::QueryStringAuthGenerator;
my $generator = Muck::FS::S3::QueryStringAuthGenerator->new(
    $access_key,
    $secret_key,
    0, # 0 means use 'http'. set this to 1 for 'https'
    'objects.dreamhost.com',
);

my $hello_url = $generator->make_bare_url($bucket->bucket, 'hello.txt');
print $hello_url . "\n";

$generator->expires_in(3600); # 1 hour = 3600 seconds
my $plans_url = $generator->get($bucket->bucket, 'secret_plans.txt');
print $plans_url . "\n";

```

The output will look something like this:

```

http://objects.dreamhost.com:80/my-bucket-name/hello.txt
http://objects.dreamhost.com:80/my-bucket-name/secret_plans.txt?Signature=XXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

## PHP S3 Examples

### Creating a Connection

This creates a connection so that you can interact with the server.

```

<?php
define('AWS_KEY', 'place access key here');
define('AWS_SECRET_KEY', 'place secret key here');
define('AWS_CANONICAL_ID', 'your DHO Username');
define('AWS_CANONICAL_NAME', 'Also your DHO Username!');
$HOST = 'objects.dreamhost.com';

// require the amazon sdk for php library
require_once 'AWSSDKforPHP/sdk.class.php';

// Instantiate the S3 class and point it at the desired host
$Connection = new AmazonS3();
$Connection->set_hostname($HOST);
$Connection->allow_hostname_override(false);

// Set the S3 class to use objects.dreamhost.com/bucket
// instead of bucket.objects.dreamhost.com
$Connection->enable_path_style();

```

### Listing Owned Buckets

This gets a list of CFSimpleXML objects representing buckets that you own. This also prints out the bucket name and creation date of each bucket.

```

<?php
$ListResponse = $Connection->list_buckets();
$Buckets = $ListResponse->body->Buckets->Bucket;
foreach ($Buckets as $Bucket) {

```

```
        echo $Bucket->Name . "\t" . $Bucket->CreationDate . "\n";
    }
```

The output will look something like this:

```
mahbucket1    2011-04-21T18:05:39.000Z
mahbucket2    2011-04-21T18:05:48.000Z
mahbucket3    2011-04-21T18:07:18.000Z
```

### Creating a Bucket

This creates a new bucket called `my-new-bucket` and returns a `CFResponse` object.

---

**Note:** This command requires a region as the second argument, so we use `AmazonS3::REGION_US_E1`, because this constant is "

---

```
<?php
$Connection->create_bucket('my-new-bucket', AmazonS3::REGION_US_E1);
```

### List a Bucket's Content

This gets an array of `CFSimpleXML` objects representing the objects in the bucket. This then prints out each object's name, the file size, and last modified date.

```
<?php
$ObjectsListResponse = $Connection->list_objects($bucketname);
$Objects = $ObjectsListResponse->body->Contents;
foreach ($Objects as $Object) {
    echo $Object->Key . "\t" . $Object->Size . "\t" . $Object->LastModified . "\n";
}
```

---

**Note:** If there are more than 1000 objects in this bucket, you need to check `$ObjectListResponse->body->isTruncated` and run again with the name of the last key listed. Keep doing this until `isTruncated` is not true.

---

The output will look something like this if the bucket has some files:

```
myphoto1.jpg 251262 2011-08-08T21:35:48.000Z
myphoto2.jpg 262518 2011-08-08T21:38:01.000Z
```

### Deleting a Bucket

This deletes the bucket called `my-old-bucket` and returns a `CFResponse` object

---

**Note:** The Bucket must be empty! Otherwise it won't work!

---

```
<?php
$Connection->delete_bucket('my-old-bucket');
```

### Forced Delete for Non-empty Buckets

This will delete the bucket even if it is not empty.

```
<?php
$Connection->delete_bucket('my-old-bucket', 1);
```

### Creating an Object

This creates an object `hello.txt` with the string "Hello World!"

```
<?php
$Connection->create_object('my-bucket-name', 'hello.txt', array(
    'body' => "Hello World!",
));
```

### Change an Object's ACL

This makes the object `hello.txt` to be publicly readable and `secret_plans.txt` to be private.

```
<?php
$Connection->set_object_acl('my-bucket-name', 'hello.txt', AmazonS3::ACL_PUBLIC);
$Connection->set_object_acl('my-bucket-name', 'secret_plans.txt', AmazonS3::ACL_PRIVATE);
```

### Delete an Object

This deletes the object `goodbye.txt`

```
<?php
$Connection->delete_object('my-bucket-name', 'goodbye.txt');
```

### Download an Object (to a file)

This downloads the object `poetry.pdf` and saves it in `/home/larry/documents/`

```
<?php
$FileHandle = fopen('/home/larry/documents/poetry.pdf', 'w+');
$Connection->get_object('my-bucket-name', 'poetry.pdf', array(
    'fileDownload' => $FileHandle,
));
```

### Generate Object Download URLs (signed and unsigned)

This generates an unsigned download URL for `hello.txt`. This works because we made `hello.txt` public by setting the ACL above. This then generates a signed download URL for `secret_plans.txt` that will work for 1 hour. Signed download URLs will work for the time period even if the object is private (when the time period is up, the URL will stop working).

```
<?php
my $plans_url = $Connection->get_object_url('my-bucket-name', 'hello.txt');
echo $plans_url . "\n";
my $secret_url = $Connection->get_object_url('my-bucket-name', 'secret_plans.txt', '1 hour');
echo $secret_url . "\n";
```

The output of this will look something like:

```
http://objects.dreamhost.com/my-bucket-name/hello.txt
http://objects.dreamhost.com/my-bucket-name/secret_plans.txt?Signature=XXXXXXXXXXXXXXXXXXXXXXXXXXXX&E
```

## Python S3 Examples

### Creating a Connection

This creates a connection so that you can interact with the server.

```
import boto
import boto.s3.connection
access_key = 'put your access key here!'
secret_key = 'put your secret key here!'

conn = boto.connect_s3(
    aws_access_key_id = access_key,
    aws_secret_access_key = secret_key,
    host = 'objects.dreamhost.com',
    calling_format = boto.s3.connection.OrdinaryCallingFormat(),
)
```

### Listing Owned Buckets

This gets a list of Buckets that you own. This also prints out the bucket name and creation date of each bucket.

```
for bucket in conn.get_all_buckets():
    print "{name}\t{created}".format(
        name = bucket.name,
        created = bucket.creation_date,
    )
```

The output will look something like this:

```
mahbuckat1    2011-04-21T18:05:39.000Z
mahbuckat2    2011-04-21T18:05:48.000Z
mahbuckat3    2011-04-21T18:07:18.000Z
```

### Creating a Bucket

This creates a new bucket called my-new-bucket

```
bucket = conn.create_bucket('my-new-bucket')
```

## Listing a Bucket's Content

This gets a list of objects in the bucket. This also prints out each object's name, the file size, and last modified date.

```
for key in bucket.list():
    print "{name}\t{size}\t{modified}".format(
        name = key.name,
        size = key.size,
        modified = key.last_modified,
    )
```

The output will look something like this:

```
myphoto1.jpg 251262 2011-08-08T21:35:48.000Z
myphoto2.jpg 262518 2011-08-08T21:38:01.000Z
```

## Deleting a Bucket

---

**Note:** The Bucket must be empty! Otherwise it won't work!

---

```
conn.delete_bucket(bucket.name)
```

## Forced Delete for Non-empty Buckets

**Attention:** not available in python

## Creating an Object

This creates a file `hello.txt` with the string "Hello World!"

```
key = bucket.new_key('hello.txt')
key.set_contents_from_string('Hello World!')
```

## Change an Object's ACL

This makes the object `hello.txt` to be publicly readable, and `secret_plans.txt` to be private.

```
hello_key = bucket.get_key('hello.txt')
hello_key.set_canned_acl('public-read')
plans_key = bucket.get_key('secret_plans.txt')
plans_key.set_canned_acl('private')
```

## Download an Object (to a file)

This downloads the object `perl_poetry.pdf` and saves it in `/home/larry/documents/`

```
key = bucket.get_key('perl_poetry.pdf')
key.get_contents_to_filename('/home/larry/documents/perl_poetry.pdf')
```

## Delete an Object

This deletes the object `goodbye.txt`

```
bucket.delete_key('goodbye.txt')
```

## Generate Object Download URLs (signed and unsigned)

This generates an unsigned download URL for `hello.txt`. This works because we made `hello.txt` public by setting the ACL above. This then generates a signed download URL for `secret_plans.txt` that will work for 1 hour. Signed download URLs will work for the time period even if the object is private (when the time period is up, the URL will stop working).

```
hello_key = bucket.get_key('hello.txt')
hello_url = hello_key.generate_url(0, query_auth=False, force_http=True)
print hello_url

plans_key = bucket.get_key('secret_plans.txt')
plans_url = plans_key.generate_url(3600, query_auth=True, force_http=True)
print plans_url
```

The output of this will look something like:

```
http://objects.dreamhost.com/my-bucket-name/hello.txt
http://objects.dreamhost.com/my-bucket-name/secret_plans.txt?Signature=XXXXXXXXXXXXXXXXXXXXXXXXXXXX&Expires=1234567890
```

## Ruby S3 Examples

### Creating a Connection

This creates a connection so that you can interact with the server.

```
AWS::S3::Base.establish_connection!(
  :server          => 'objects.dreamhost.com',
  :use_ssl         => true,
  :access_key_id   => 'my-access-key',
  :secret_access_key => 'my-secret-key'
)
```

### Listing Owned Buckets

This gets a list of `AWS::S3::Bucket` objects that you own. This also prints out the bucket name and creation date of each bucket.

```
AWS::S3::Service.buckets.each do |bucket|
  puts "#{bucket.name}\t#{bucket.creation_date}"
end
```

The output will look something like this:

```
mahbuckat1    2011-04-21T18:05:39.000Z
mahbuckat2    2011-04-21T18:05:48.000Z
mahbuckat3    2011-04-21T18:07:18.000Z
```

## Creating a Bucket

This creates a new bucket called `my-new-bucket`

```
AWS::S3::Bucket.create('my-new-bucket')
```

## Listing a Bucket's Content

This gets a list of hashes with the contents of each object This also prints out each object's name, the file size, and last modified date.

```
new_bucket = AWS::S3::Bucket.find('my-new-bucket')
new_bucket.each do |object|
  puts "#{object.key}\t#{object.about['content-length']}\t#{object.about['last-modified']}"
end
```

The output will look something like this if the bucket has some files:

```
myphoto1.jpg 251262 2011-08-08T21:35:48.000Z
myphoto2.jpg 262518 2011-08-08T21:38:01.000Z
```

## Deleting a Bucket

---

**Note:** The Bucket must be empty! Otherwise it won't work!

---

```
AWS::S3::Bucket.delete('my-new-bucket')
```

## Forced Delete for Non-empty Buckets

```
AWS::S3::Bucket.delete('my-new-bucket', :force => true)
```

## Creating an Object

This creates a file `hello.txt` with the string `"Hello World!"`

```
AWS::S3::S3Object.store(
  'hello.txt',
  'Hello World!',
  'my-new-bucket',
  :content_type => 'text/plain'
)
```

## Change an Object's ACL

This makes the object `hello.txt` to be publicly readable, and `secret_plans.txt` to be private.

```
policy = AWS::S3::S3Object.acl('hello.txt', 'my-new-bucket')
policy.grants = [ AWS::S3::ACL::Grant.grant(:public_read) ]
AWS::S3::S3Object.acl('hello.txt', 'my-new-bucket', policy)
```

```
policy = AWS::S3::S3Object.acl('secret_plans.txt', 'my-new-bucket')
policy.grants = []
AWS::S3::S3Object.acl('secret_plans.txt', 'my-new-bucket', policy)
```

### Download an Object (to a file)

This downloads the object `poetry.pdf` and saves it in `/home/larry/documents/`

```
open('/home/larry/documents/poetry.pdf', 'w') do |file|
  AWS::S3::S3Object.stream('poetry.pdf', 'my-new-bucket') do |chunk|
    file.write(chunk)
  end
end
```

### Delete an Object

This deletes the object `goodbye.txt`

```
AWS::S3::S3Object.delete('goodbye.txt', 'my-new-bucket')
```

### Generate Object Download URLs (signed and unsigned)

This generates an unsigned download URL for `hello.txt`. This works because we made `hello.txt` public by setting the ACL above. This then generates a signed download URL for `secret_plans.txt` that will work for 1 hour. Signed download URLs will work for the time period even if the object is private (when the time period is up, the URL will stop working).

```
puts AWS::S3::S3Object.url_for(
  'hello.txt',
  'my-new-bucket',
  :authenticated => false
)

puts AWS::S3::S3Object.url_for(
  'secret_plans.txt',
  'my-new-bucket',
  :expires_in => 60 * 60
)
```

The output of this will look something like:

```
http://objects.dreamhost.com/my-bucket-name/hello.txt
http://objects.dreamhost.com/my-bucket-name/secret_plans.txt?Signature=XXXXXXXXXXXXXXXXXXXXXXXXXXXX&E
```

## 7.4.2 Features Support

The following table describes the support status for current Amazon S3 functional features:



| Feature                   | Status        | Remarks                      |
|---------------------------|---------------|------------------------------|
| List Buckets              | Supported     |                              |
| Delete Bucket             | Supported     |                              |
| Create Bucket             | Supported     | Different set of canned ACLs |
| Bucket Lifecycle          | Not Supported |                              |
| Policy (Buckets, Objects) | Not Supported | ACLs are supported           |
| Bucket Website            | Not Supported |                              |
| Bucket ACLs (Get, Put)    | Supported     | Different set of canned ACLs |
| Bucket Location           | Not Supported |                              |
| Bucket Notification       | Not Supported |                              |
| Bucket Object Versions    | Not Supported |                              |
| Get Bucket Info (HEAD)    | Supported     |                              |
| Bucket Request Payment    | Not Supported |                              |
| Put Object                | Supported     |                              |
| Delete Object             | Supported     |                              |
| Get Object                | Supported     |                              |
| Object ACLs (Get, Put)    | Supported     |                              |
| Get Object Info (HEAD)    | Supported     |                              |
| POST Object               | Not Supported |                              |
| Copy Object               | Supported     |                              |
| Multipart Uploads         | Supported     | (missing Copy Part)          |

### 7.4.3 Unsupported Header Fields

The following common request header fields are not supported:

| Name                 | Type     |
|----------------------|----------|
| x-amz-security-token | Request  |
| Server               | Response |
| x-amz-delete-marker  | Response |
| x-amz-id-2           | Response |
| x-amz-request-id     | Response |
| x-amz-version-id     | Response |

## 7.5 Swift-compatible API

RADOS Gateway provides a scalable, highly available redundant object storage API that is compatible with a subset of the [OpenStack Swift](#) API. The Swift-compatible API provides a container-based object storage, with support for multiple users, storage containers, and access control lists (ACLs). This API makes it possible to use a RADOS storage cluster as a Swift-compatible object storage system, while simultaneously supporting Ceph FS and RADOS block devices too (*e.g.*, you can use it for your Rackspace Cloud Files).

---

**Note:** The popular Amazon S3 API uses the term ‘bucket’ to describe a data container. When you hear someone refer to a ‘bucket’ within the Swift API, the term ‘bucket’ may be construed as the equivalent of the term ‘container.’

---

### 7.5.1 Tutorial

The Swift-compatible API tutorials follow a simple container-based object lifecycle. The first step requires you to setup a connection between your client and the RADOS Gateway server. Then, you may follow a natural container

and object lifecycle, including adding and retrieving object metadata. See example code for the following languages:

- Java
- Python
- Ruby

## 7.5.2 Java Swift Examples

### Setup

The following examples may require some or all of the following Java classes to be imported:

```
import java.io.File;
import java.util.List;
import java.util.Map;
import com.rackspacecloud.client.cloudfiles.FilesClient;
import com.rackspacecloud.client.cloudfiles.FilesConstants;
import com.rackspacecloud.client.cloudfiles.FilesContainer;
import com.rackspacecloud.client.cloudfiles.FilesContainerExistsException;
import com.rackspacecloud.client.cloudfiles.FilesObject;
import com.rackspacecloud.client.cloudfiles.FilesObjectMetaData;
```

### Create a Connection

This creates a connection so that you can interact with the server:

```
String username = "USERNAME";
String password = "PASSWORD";
String authUrl = "https://objects.dreamhost.com/auth";

FilesClient client = new FilesClient(username, password, authUrl);
if (!client.login()) {
    throw new RuntimeException("Failed to log in");
}
```

### Create a Container

This creates a new container called my-new-container:

```
client.createContainer("my-new-container");
```

### Create an Object

This creates an object foo.txt from the file named foo.txt in the container my-new-container:

```
File file = new File("foo.txt");
String mimeType = FilesConstants.getMimeType("txt");
client.storeObject("my-new-container", file, mimeType);
```

## Add/Update Object Metadata

This adds the metadata key-value pair `key:value` to the object named `foo.txt` in the container `my-new-container`:

```
FilesObjectMetaData metaData = client.getObjectMetaData("my-new-container", "foo.txt");
metaData.addMetaData("key", "value");

Map<String, String> metamap = metaData.getMetaData();
client.updateObjectMetadata("my-new-container", "foo.txt", metamap);
```

## List Owned Containers

This gets a list of Containers that you own. This also prints out the container name.

```
List<FilesContainer> containers = client.listContainers();
for (FilesContainer container : containers) {
    System.out.println(" " + container.getName());
}
```

The output will look something like this:

```
mahbuckat1
mahbuckat2
mahbuckat3
```

## List a Container's Content

This gets a list of objects in the container `my-new-container`; and, it also prints out each object's name, the file size, and last modified date:

```
List<FilesObject> objects = client.listObjects("my-new-container");
for (FilesObject object : objects) {
    System.out.println(" " + object.getName());
}
```

The output will look something like this:

```
myphoto1.jpg
myphoto2.jpg
```

## Retrieve an Object's Metadata

This retrieves metadata and gets the MIME type for an object named `foo.txt` in a container named `my-new-container`:

```
FilesObjectMetaData metaData = client.getObjectMetaData("my-new-container", "foo.txt");
String mimeType = metaData.getMimeType();
```

## Retrieve an Object

This downloads the object `foo.txt` in the container `my-new-container` and saves it in `./outfile.txt`:

```
FilesObject obj;
File outfile = new File("outfile.txt");

List<FilesObject> objects = client.listObjects("my-new-container");
for (FilesObject object : objects) {
    String name = object.getName();
    if (name.equals("foo.txt")) {
        obj = object;
        obj.writeObjectToFile(outfile);
    }
}
```

### Delete an Object

This deletes the object `goodbye.txt` in the container “my-new-container”:

```
client.deleteObject("my-new-container", "goodbye.txt");
```

### Delete a Container

This deletes a container named “my-new-container”:

```
client.deleteContainer("my-new-container");
```

---

**Note:** The container must be empty! Otherwise it won’t work!

---

## 7.5.3 Python Swift Examples

### Create a Connection

This creates a connection so that you can interact with the server:

```
import cloudfiles
username = 'account_name:username'
api_key = 'your_api_key'

conn = cloudfiles.get_connection(
    username=username,
    api_key=api_key,
    authurl='https://objects.dreamhost.com/auth',
)
```

### Create a Container

This creates a new container called `my-new-container`:

```
container = conn.create_container('my-new-container')
```

## Create an Object

This creates a file `hello.txt` from the file named `my_hello.txt`:

```
obj = container.create_object('hello.txt')
obj.content_type = 'text/plain'
obj.load_from_filename('./my_hello.txt')
```

## List Owned Containers

This gets a list of containers that you own, and prints out the container name:

```
for container in conn.get_all_containers():
    print container.name
```

The output will look something like this:

```
mahbuckat1
mahbuckat2
mahbuckat3
```

## List a Container's Content

This gets a list of objects in the container, and prints out each object's name, the file size, and last modified date:

```
for obj in container.get_objects():
    print "{0}\t{1}\t{2}".format(obj.name, obj.size, obj.last_modified)
```

The output will look something like this:

```
myphoto1.jpg 251262 2011-08-08T21:35:48.000Z
myphoto2.jpg 262518 2011-08-08T21:38:01.000Z
```

## Retrieve an Object

This downloads the object `hello.txt` and saves it in `./my_hello.txt`:

```
obj = container.get_object('hello.txt')
obj.save_to_filename('./my_hello.txt')
```

## Delete an Object

This deletes the object `goodbye.txt`:

```
container.delete_object('goodbye.txt')
```

## Delete a Container

---

**Note:** The container must be empty! Otherwise the request won't work!

---

```
conn.delete_container(container.name)
```

## 7.5.4 Ruby Swift Examples

### Create a Connection

This creates a connection so that you can interact with the server:

```
require 'cloudfiles'
username = 'account_name:user_name'
api_key   = 'your_secret_key'

conn = CloudFiles::Connection.new(
  :username => username,
  :api_key  => api_key,
  :auth_url => 'http://objects.dreamhost.com/auth'
)
```

### Create a Container

This creates a new container called `my-new-container`

```
container = conn.create_container('my-new-container')
```

### Create an Object

This creates a file `hello.txt` from the file named `my_hello.txt`

```
obj = container.create_object('hello.txt')
obj.load_from_filename('./my_hello.txt')
obj.content_type = 'text/plain'
```

### List Owned Containers

This gets a list of Containers that you own, and also prints out the container name:

```
conn.containers.each do |container|
  puts container
end
```

The output will look something like this:

```
mahbuckat1
mahbuckat2
mahbuckat3
```

### List a Container's Contents

This gets a list of objects in the container, and prints out each object's name, the file size, and last modified date:

```
require 'date' # not necessary in the next version

container.objects_detail.each do |name, data|
  puts "#{name}\\t#{data[:bytes]}\\t#{data[:last_modified]}"
end
```

The output will look something like this:

```
myphoto1.jpg 251262 2011-08-08T21:35:48.000Z
myphoto2.jpg 262518 2011-08-08T21:38:01.000Z
```

## Retrieve an Object

This downloads the object `hello.txt` and saves it in `./my_hello.txt`:

```
obj = container.object('hello.txt')
obj.save_to_filename('./my_hello.txt')
```

## Delete an Object

This deletes the object `goodbye.txt`:

```
container.delete_object('goodbye.txt')
```

## Delete a Container

---

**Note:** The container must be empty! Otherwise the request won't work!

---

```
container.delete_container('my-new-container')
```

## 7.5.5 Authentication

Swift API requests that require authentication must contain an X-Storage-Token authentication token in the request header. The token may be retrieved from RADOS Gateway, or from another authenticator. To obtain a token from RADOS Gateway, you must create a user. For example:

```
sudo radosgw-admin user create --uid="{username}" --displayname="{Display Name}"
```

For details on RADOS Gateway administration, see `radosgw-admin`.

### Auth Get

To authenticate a user, make a request containing an X-Auth-User and a X-Auth-Key in the header.

### Syntax

```
GET /auth HTTP/1.1
Host: swift.radosgw.localhost
X-Auth-User: johndoe
X-Auth-Key: R7UUOLF2I2ZI9PRCQ53K
```

## Request Headers

X-Auth-User

**Description** The key RADOS GW username to authenticate.

**Type** String

**Required** Yes

X-Auth-Key

**Description** The key associated to a RADOS GW username.

**Type** String

**Required** Yes

## Response Headers

The response from the server should include an X-Auth-Token value. The response may also contain a X-Storage-Url that provides the {api version}/{account} prefix that is specified in other requests throughout the API documentation.

X-Storage-Token

**Description** The authorization token for the X-Auth-User specified in the request.

**Type** String

X-Storage-Url

**Description** The URL and {api version}/{account} path for the user.

**Type** String

A typical response looks like this:

```
HTTP/1.1 204 No Content
Date: Mon, 16 Jul 2012 11:05:33 GMT
Server: swift
X-Storage-Url: https://swift.radosgwhost.com/v1/ACCT-12345
X-Auth-Token: U0lCCC8TahFKlWuv9DB09TWHF0nDjpPElha0kAa
Content-Length: 0
Content-Type: text/plain; charset=UTF-8
```

## 7.5.6 Service Operations

To retrieve data about our Swift-compatible service, you may execute GET requests using the X-Storage-Url value retrieved during authentication.

### List Containers

A GET request that specifies the API version and the account will return a list of containers for a particular user account. Since the request returns a particular user's containers, the request requires an authentication token. The request cannot be made anonymously.



## Syntax

```
GET /{api version}/{account} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

## Request Parameters

limit

**Description** Limits the number of results to the specified value.

**Type** Integer

**Required** No

format

**Description** Defines the format of the result.

**Type** String

**Valid Values** json|xml

**Required** No

marker

**Description** Returns a list of results greater than the marker value.

**Type** String

**Required** No

## Response Entities

The response contains a list of containers, or returns with an HTTP 204 response code

account

**Description** A list for account information.

**Type** Container

container

**Description** The list of containers.

**Type** Container

name

**Description** The name of a container.

**Type** String

bytes

**Description** The size of the container.

**Type** Integer

## 7.5.7 Container Operations

A container is a mechanism for storing data objects. An account may have many containers, but container names must be unique. This API enables a client to create a container, set access controls and metadata, retrieve a container's contents, and delete a container. Since this API makes requests related to information in a particular user's account, all requests in this API must be authenticated unless a container's access control is deliberately made publicly accessible (i.e., allows anonymous requests).

---

**Note:** The Amazon S3 API uses the term 'bucket' to describe a data container. When you hear someone refer to a 'bucket' within the Swift API, the term 'bucket' may be construed as the equivalent of the term 'container.'

---

One facet of object storage is that it does not support hierarchical paths or directories. Instead, it supports one level consisting of one or more containers, where each container may have objects. The RADOS Gateway's Swift-compatible API supports the notion of 'pseudo-hierarchical containers,' which is a means of using object naming to emulate a container (or directory) hierarchy without actually implementing one in the storage system. You may name objects with pseudo-hierarchical names (e.g., photos/buildings/empire-state.jpg), but container names cannot contain a forward slash (/) character.

### Create a Container

To create a new container, make a PUT request with the API version, account, and the name of the new container. The container name must be unique, must not contain a forward-slash (/) character, and should be less than 256 bytes. You may include access control headers and metadata headers in the request. The operation is idempotent; that is, if you make a request to create a container that already exists, it will return with a HTTP 202 return code, but will not create another container.

### Syntax

```
PUT /{api version}/{account}/{container} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
X-Container-Read: {comma-separated-uids}
X-Container-Write: {comma-separated-uids}
X-Container-Meta-{key}: {value}
```

### Headers

X-Container-Read

**Description** The user IDs with read permissions for the container.

**Type** Comma-separated string values of user IDs.

**Required** No

X-Container-Write

**Description** The user IDs with write permissions for the container.

**Type** Comma-separated string values of user IDs.

**Required** No

X-Container-Meta-{key}

**Description** A user-defined meta data key that takes an arbitrary string value.

**Type** String

**Required** No

### HTTP Response

If a container with the same name already exists, and the user is the container owner then the operation will succeed. Otherwise the operation will fail.

409

**Description** The container already exists under a different user's ownership.

**Status Code** `BucketAlreadyExists`

### List a Container's Objects

To list the objects within a container, make a GET request with the with the API version, account, and the name of the container. You can specify query parameters to filter the full list, or leave out the parameters to return a list of the first 10,000 object names stored in the container.

### Syntax

```
GET /{api version}/{container} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

### Parameters

`format`

**Description** Defines the format of the result.

**Type** String

**Valid Values** `json|xml`

**Required** No

`prefix`

**Description** Limits the result set to objects beginning with the specified prefix.

**Type** String

**Required** No

`marker`

**Description** Returns a list of results greater than the marker value.

**Type** String

**Required** No

`limit`

**Description** Limits the number of results to the specified value.

**Type** Integer

**Valid Range** 0 - 10,000

**Required** No

delimiter

**Description** The delimiter between the prefix and the rest of the object name.

**Type** String

**Required** No

path

**Description** The pseudo-hierarchical path of the objects.

**Type** String

**Required** No

## Response Entities

container

**Description** The container.

**Type** Container

object

**Description** An object within the container.

**Type** Container

name

**Description** The name of an object within the container.

**Type** String

hash

**Description** A hash code of the object's contents.

**Type** String

last\_modified

**Description** The last time the object's contents were modified.

**Type** Date

content\_type

**Description** The type of content within the object.

**Type** String

## Update a Container's ACLs

When a user creates a container, the user has read and write access to the container by default. To allow other users to read a container's contents or write to a container, you must specifically enable the user. You may also specify \* in the X-Container-Read or X-Container-Write settings, which effectively enables all users to either read

from or write to the container. Setting `*` makes the container public. That is it enables anonymous users to either read from or write to the container.

### Syntax

```
POST /{api version}/{account}/{container} HTTP/1.1
Host: {fqdn}
  X-Auth-Token: {auth-token}
  X-Container-Read: *
  X-Container-Write: {uid1}, {uid2}, {uid3}
```

### Request Headers

X-Container-Read

**Description** The user IDs with read permissions for the container.

**Type** Comma-separated string values of user IDs.

**Required** No

X-Container-Write

**Description** The user IDs with write permissions for the container.

**Type** Comma-separated string values of user IDs.

**Required** No

### Add/Update Container Metadata

To add metadata to a container, make a `POST` request with the API version, account, and container name. You must have write permissions on the container to add or update metadata.

### Syntax

```
POST /{api version}/{account}/{container} HTTP/1.1
Host: {fqdn}
  X-Auth-Token: {auth-token}
  X-Container-Meta-Color: red
  X-Container-Meta-Taste: salty
```

### Request Headers

X-Container-Meta-{key}

**Description** A user-defined meta data key that takes an arbitrary string value.

**Type** String

**Required** No

## Delete a Container

To delete a container, make a `DELETE` request with the API version, account, and the name of the container. The container must be empty. If you'd like to check if the container is empty, execute a `HEAD` request against the container. Once you've successfully removed the container, you'll be able to reuse the container name.

### Syntax

```
DELETE /{api version}/{account}/{container} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

### HTTP Response

204

**Description** The container was removed.

**Status Code** NoContent

## 7.5.8 Object Operations

An object is a container for storing data and metadata. A container may have many objects, but the object names must be unique. This API enables a client to create an object, set access controls and metadata, retrieve an object's data and metadata, and delete an object. Since this API makes requests related to information in a particular user's account, all requests in this API must be authenticated unless the container or object's access control is deliberately made publicly accessible (i.e., allows anonymous requests).

### Create/Update an Object

To create a new object, make a `PUT` request with the API version, account, container name and the name of the new object. You must have write permission on the container to create or update an object. The object name must be unique within the container. The `PUT` request is not idempotent, so if you do not use a unique name, the request will update the object. However, you may use pseudo-hierarchical syntax in your object name to distinguish it from another object of the same name if it is under a different pseudo-hierarchical directory. You may include access control headers and metadata headers in the request.

### Syntax

```
PUT /{api version}/{account}/{container}/{object} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

### Request Headers

ETag

**Description** An MD5 hash of the object's contents. Recommended.

**Type** String

**Required** No

Content-Type

**Description** The type of content the object contains.**Type** String**Required** No

Transfer-Encoding

**Description** Indicates whether the object is part of a larger aggregate object.**Type** String**Valid Values** chunked**Required** No

## Copy an Object

Copying an object allows you to make a server-side copy of an object, so that you don't have to download it and upload it under another container/name. To copy the contents of one object to another object, you may make either a PUT request or a COPY request with the API version, account, and the container name. For a PUT request, use the destination container and object name in the request, and the source container and object in the request header. For a Copy request, use the source container and object in the request, and the destination container and object in the request header. You must have write permission on the container to copy an object. The destination object name must be unique within the container. The request is not idempotent, so if you do not use a unique name, the request will update the destination object. However, you may use pseudo-hierarchical syntax in your object name to distinguish the destination object from the source object of the same name if it is under a different pseudo-hierarchical directory. You may include access control headers and metadata headers in the request.

## Syntax

```
PUT /{api version}/{account}/{dest-container}/{dest-object} HTTP/1.1
X-Copy-From: {source-container}/{source-object}
Host: {fqdn}
X-Auth-Token: {auth-token}
```

or alternatively:

```
COPY /{api version}/{account}/{source-container}/{source-object} HTTP/1.1
Destination: {dest-container}/{dest-object}
```

## Request Headers

X-Copy-From

**Description** Used with a PUT request to define the source container/object path.**Type** String**Required** Yes, if using PUT

Destination

**Description** Used with a COPY request to define the destination container/object path.**Type** String

**Required** Yes, if using COPY

If-Modified-Since

**Description** Only copies if modified since the date/time of the source object's `last_modified` attribute.

**Type** Date

**Required** No

If-Unmodified-Since

**Description** Only copies if not modified since the date/time of the source object's `last_modified` attribute.

**Type** Date

**Required** No

Copy-If-Match

**Description** Copies only if the ETag in the request matches the source object's ETag.

**Type** ETag.

**Required** No

Copy-If-None-Match

**Description** Copies only if the ETag in the request does not match the source object's ETag.

**Type** ETag.

**Required** No

## Delete an Object

To delete an object, make a `DELETE` request with the API version, account, container and object name. You must have write permissions on the container to delete an object within it. Once you've successfully deleted the object, you'll be able to reuse the object name.

### Syntax

```
DELETE /{api version}/{account}/{container}/{object} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

## Get an Object

To retrieve an object, make a `GET` request with the API version, account, container and object name. You must have read permissions on the container to retrieve an object within it.

### Syntax

```
GET /{api version}/{account}/{container}/{object} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```



## Request Headers

range

**Description** To retrieve a subset of an object's contents, you may specify a byte range.

**Type** Date

**Required** No

If-Modified-Since

**Description** Only copies if modified since the date/time of the source object's `last_modified` attribute.

**Type** Date

**Required** No

If-Unmodified-Since

**Description** Only copies if not modified since the date/time of the source object's `last_modified` attribute.

**Type** Date

**Required** No

Copy-If-Match

**Description** Copies only if the ETag in the request matches the source object's ETag.

**Type** ETag.

**Required** No

Copy-If-None-Match

**Description** Copies only if the ETag in the request does not match the source object's ETag.

**Type** ETag.

**Required** No

## Response Headers

Content-Range

**Description** The range of the subset of object contents. Returned only if the range header field was specified in the request

## Get Object Metadata

To retrieve an object's metadata, make a `HEAD` request with the API version, account, container and object name. You must have read permissions on the container to retrieve metadata from an object within the container. This request returns the same header information as the request for the object itself, but it does not return the object's data.

## Syntax

```
HEAD /{api version}/{account}/{container}/{object} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

## Add/Update Object Metadata

To add metadata to an object, make a POST request with the API version, account, container and object name. You must have write permissions on the parent container to add or update metadata.

### Syntax

```
POST /{api version}/{account}/{container}/{object} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

### Request Headers

X-Container-Meta-{key}

**Description** A user-defined meta data key that takes an arbitrary string value.

**Type** String

**Required** No

# OPERATIONS

## 8.1 Managing a Ceph cluster

### 8.1.1 Managing crypto keys

#### Types of keys

---

##### Todo

client, osd, mds, mon; id, no id

---

#### Capabilities

##### Adding a new key

##### Setting capabilities for a key

##### Revoking a key

### 8.1.2 Growing or shrinking a Ceph cluster

---

##### Todo

write me

---

#### Resizing the RADOS cluster

##### Adding a new OSD to the cluster

Briefly...

1. Allocate a new OSD id:

```
$ ceph osd create  
123
```

2. Make sure ceph.conf is valid for the new OSD.

3. Initialize osd data directory:

```
$ ceph-osd -i 123 --mkfs --mkkey
```

4. Register the OSD authentication key:

```
$ ceph auth add osd.123 osd 'allow *' mon 'allow rwx' -i /var/lib/ceph/osd-data/123/keyring
```

5. Adjust the CRUSH map to allocate data to the new device (see [Adjusting the CRUSH map](#)).

## Removing OSDs

Briefly...

1. Stop the daemon
2. Remove it from the CRUSH map:

```
$ ceph osd crush remove osd.123
```

3. Remove it from the osd map:

```
$ ceph osd rm 123
```

See also [Recovering from failures](#).

## Tuning Placement Groups

### Purpose

Placement groups (PGs) are shards or fragments of a logical object pool. Their function is to spread the responsibility for storing objects across the nodes of the cluster. Generally speaking, a large number of placement groups will ensure that load is spread evenly, at the expensive of tracking/management overhead. A small number of placement groups can result in non-optimal distribution of load (e.g., some OSDs with significantly more or less data than the others).

Each pool as a `pg_num` property that indicates the number of PGs it is fragmented into. The total number of PG copies in the system is the sum of the `pg_num` value times the replication factor for each pool.

### Optimal total PG count

A rule of thumb is to shoot for a total PG count that is on the order of 50-100 PGs per OSD in the system. Anything within an order of magnitude of that target will do reasonably well in terms of variation between utilization. Thus, if your system have 900 OSDs, you probably want somewhere between 30,000 and 90,000 PG copies. If your replication factor is 3x, then you want your `pg_num` values to add to something between 10,000 and 30,000. 16,384 is a nice round number (powers of two are marginally better).

If your system tends to be underpowered, choosing a lower `pg_num` will ease system load at the expense of load balance.

If you expect the cluster to grow in size, you may want to choose a larger `pg_num` that will remain within a good range as the number of OSDs increases.

## Multiple pools

If you have a single pool of objects in the system, choosing `pg_num` is easy:

```
total_pg_copies = num_osds * 100
total_pgs = total_pg_copies / replication_factor
lone_pool_pg_num = total_pgs
```

If you have multiple pools, you need to be a bit careful about how each pool's `pg_num` value is chosen so that they add up to a reasonable total. Generally speaking, each pool's `pg_num` should be roughly proportional to the number of objects you expect the pool to contain.

## Splitting/merging PGs

Ceph will soon support the ability to dynamically adjust the number of PGs in a pool after it has been created. Existing PGs will be “split” into smaller fragments or “merged” into larger ones and then redistributed. Until then, the `pg_num` pool property can only be set when the pool is created.

## Resizing the metadata cluster

### Adding new MDSes

### Setting up standby and standby-replay MDSes

### Removing MDSes

#### Status as of 2011-09:

You can remove an MDS from the system by executing “`ceph mds deactivate x`”, where `x` is numerical ID of the MDS to shut down. Beware: shrinking the number of MDSes is not well tested.

## Resizing the monitor cluster

### Adding a monitor

1. Initialize the new monitor's data directory with the `ceph-mon --mkfs` command. You need to provide the new monitor with three pieces of information:
  - the cluster `fsid`. This can come from a `monmap` (`--monmap </path/to/monmap>`) or explicitly via `--fsid <fsid>`.
  - one or more existing monitors to join. This can come via `-m <host1,host2,...>`, a `monmap` (`--monmap </some/path>`), or `[mon.foo]` sections with `mon addr` fields in `ceph.conf`.
  - the monitor authentication key `mon..`. This should be passed in explicitly via a keyring (`--keyring </some/path>`).

Any combination of the above arguments that provide the four needed pieces of information will work. The simplest way to do this is usually:

```
$ ceph mon getmap -o /tmp/monmap          # provides fsid and existing monitor addrs
$ ceph auth export mon. -o /tmp/monkey    # mon. auth key
$ ceph-mon -i newname --mkfs --monmap /tmp/monmap --keyring /tmp/monkey
```

2. Start the new monitor and it will automatically join the cluster. The daemon needs to know which address to bind to, either via `--public-addr <ip:port>` or by setting `mon addr` in the appropriate section of *ceph.conf*. For example:

```
$ ceph-mon -i newname --public-addr <ip:port>
```

3. If you would like other nodes to be able to use this monitor during their initial startup, you'll need to adjust *ceph.conf* to add a section and `mon addr` for the new monitor, or add it to the existing `mon host` list.

### Removing a monitor from a healthy cluster

If the cluster is healthy, you can do:

```
$ ceph mon remove $id
```

For example, if your cluster includes `mon.a`, `mon.b`, and `mon.c`, then you can remove `mon.c` with:

```
$ ceph mon remove c
```

### Removing a monitor from an unhealthy or down cluster

The mon cluster may not be up because you have lost too many nodes to form a quorum.

1. On a surviving monitor node, find the most recent monmap:

```
$ ls $mon_data/monmap
1 2  accepted_pn  last_committed  latest
```

in this case it is 2.

2. Copy to a temporary location and modify the monmap to remove the node(s) you don't want. Let's say the map has `mon.a`, `mon.b`, and `mon.c`, but only `mon.a` is surviving:

```
$ cp $mon_data/monmap/2 /tmp/foo
$ monmaptool /tmp/foo --rm b
$ monmaptool /tmp/foo --rm c
```

3. Make sure `ceph-mon` isn't running:

```
$ service ceph stop mon
```

4. Inject the modified map on any surviving nodes. For example, for `mon.a`:

```
$ ceph-mon -i a --inject-monmap /tmp/foo # for each surviving monitor
```

5. Start the surviving monitor(s):

```
$ service ceph start mon # on each node with a surviving monitor
```

6. Remove the old monitors from *ceph.conf* so that nobody tries to connect to the old instances.

### 8.1.3 Recovering from failures

The current health of the Ceph cluster, as known by the monitors, can be checked with the `ceph health` command. If all is well, you get:

```
$ ceph health
HEALTH_OK
```

If there are problems, you will see something like:

```
$ ceph health
HEALTH_WARN short summary of problem(s)
```

or:

```
$ ceph health
HEALTH_ERROR short summary of very serious problem(s)
```

To get more detail:

```
$ ceph health detail
HEALTH_WARN short description of problem
```

```
one problem
another problem
yet another problem
...
```

#### Recovering from ceph-mon failure

Any single ceph-mon failure should not take down the entire monitor cluster as long as a majority of the nodes are available. If that is the case—the remaining nodes are able to form a quorum—the `ceph health` command will report any problems:

```
$ ceph health
HEALTH_WARN 1 mons down, quorum 0,2
```

and:

```
$ ceph health detail
HEALTH_WARN 1 mons down, quorum 0,2
mon.b (rank 1) addr 192.168.106.220:6790/0 is down (out of quorum)
```

Generally speaking, simply restarting the affected node will repair things.

If there are not enough monitors to form a quorum, the `ceph` command will block trying to reach the cluster. In this situation, you need to get enough `ceph-mon` daemons running to form a quorum before doing anything else with the cluster.

#### Replacing a monitor

If, for some reason, a monitor data store becomes corrupt, the monitor can be recreated and allowed to rejoin the cluster, much like a normal monitor cluster expansion. See [Adding a monitor](#).

## Recovering from ceph-osd failure

### Single ceph-osd failure

When a ceph-osd process dies, the monitor will learn about the failure from surviving ceph-osd daemons and report it via the `ceph health` command:

```
$ ceph health
HEALTH_WARN 1/3 in osds are down
```

Specifically, you will get a warning whenever there are ceph-osd processes that are marked in and down. You can identify which ceph-osds are down with:

```
$ ceph health detail
HEALTH_WARN 1/3 in osds are down
osd.0 is down since epoch 23, last address 192.168.106.220:6800/11080
```

Under normal circumstances, simply restarting the ceph-osd daemon will allow it to rejoin the cluster and recover. If there is a disk failure or other fault preventing ceph-osd from functioning or restarting, an error message should be present in its log file in `/var/log/ceph`.

If the daemon stopped because of a heartbeat failure, the underlying kernel file system may be unresponsive; check `dmesg` output for disk or other kernel errors.

If the problem is a software error (failed assertion or other unexpected error), it should be reported to the [mailing list](#).

### Full cluster

If the cluster fills up, the monitor will prevent new data from being written. The system puts ceph-osds in two categories: `nearfull` and `full`, with configurable thresholds for each (80% and 90% by default). In both cases, full ceph-osds will be reported by `ceph health`:

```
$ ceph health
HEALTH_WARN 1 nearfull osds
osd.2 is near full at 85%
```

or:

```
$ ceph health
HEALTH_ERR 1 nearfull osds, 1 full osds
osd.2 is near full at 85%
osd.3 is full at 97%
```

The best way to deal with a full cluster is to add new ceph-osds, allowing the cluster to redistribute data to the newly available storage.

### Homeless placement groups (PGs)

It is possible for all OSDs that had copies of a given PG to fail. If that's the case, that subset of the object store is unavailable, and the monitor will receive no status updates for those PGs. To detect this situation, the monitor marks any PG whose primary OSD has failed as *stale*. For example:

```
$ ceph health
HEALTH_WARN 24 pgs stale; 3/300 in osds are down
```

You can identify which PGs are stale, and what the last OSDs to store them were, with:



```
$ ceph health detail
HEALTH_WARN 24 pgs stale; 3/300 in osds are down
...
pg 2.5 is stuck stale+active+remapped, last acting [2,0]
...
osd.10 is down since epoch 23, last address 192.168.106.220:6800/11080
osd.11 is down since epoch 13, last address 192.168.106.220:6803/11539
osd.12 is down since epoch 24, last address 192.168.106.220:6806/11861
```

If we want to get PG 2.5 back online, for example, this tells us that it was last managed by ceph-osds 0 and 2. Restarting those ceph-osd daemons will allow the cluster to recover that PG (and, presumably, many others).

### Stuck PGs

It is normal for PGs to enter states like “degraded” or “peering” following a failure. Normally these states indicate the normal progression through the failure recovery process. However, if a PG stays in one of these states for a long time this may be an indication of a larger problem. For this reason, the monitor will warn when PGs get “stuck” in a non-optimal state. Specifically, we check for:

- `inactive` - the PG has not been active for too long (i.e., hasn’t been able to service read/write requests)
- `unclean` - the PG has not been `clean` for too long (i.e., hasn’t been able to completely recover from a previous failure)
- `stale` - the PG status has not been updated by a `ceph-osd`, indicating that all nodes storing this PG may be down

You can explicitly list stuck PGs with one of:

```
$ ceph pg dump_stuck stale
$ ceph pg dump_stuck inactive
$ ceph pg dump_stuck unclean
```

For stuck stale PGs, it is normally a matter of getting the right ceph-osd daemons running again. For stuck inactive PGs, it is usually a peering problem (see [PG down \(peering failure\)](#)). For stuck unclean PGs, there is usually something preventing recovery from completing, like unfound objects (see [Unfound objects](#));

### PG down (peering failure)

In certain cases, the ceph-osd “peering” process can run into problems, preventing a PG from becoming active and usable. For example, `ceph health` might report:

```
$ ceph health detail
HEALTH_ERR 7 pgs degraded; 12 pgs down; 12 pgs peering; 1 pgs recovering; 6 pgs stuck unclean; 114/300
...
pg 0.5 is down+peering
pg 1.4 is down+peering
...
osd.1 is down since epoch 69, last address 192.168.106.220:6801/8651
```

We can query the cluster to determine exactly why the PG is marked down with:

```
$ ceph pg 0.5 query
{ "state": "down+peering",
  ...
  "recovery_state": [
    { "name": "Started\\Primary\\Peering\\GetInfo",
```

```
"enter_time": "2012-03-06 14:40:16.169679",
"requested_info_from": [],
{ "name": "Started\\Primary\\Peering",
  "enter_time": "2012-03-06 14:40:16.169659",
  "probing_osds": [
    0,
    1],
  "blocked": "peering is blocked due to down osds",
  "down_osds_we_would_probe": [
    1],
  "peering_blocked_by": [
    { "osd": 1,
      "current_lost_at": 0,
      "comment": "starting or marking this osd lost may let us proceed" }],
{ "name": "Started",
  "enter_time": "2012-03-06 14:40:16.169513" } ] }
```

The `recovery_state` section tells us that peering is blocked due to down ceph-osd daemons, specifically `osd.1`. In this case, we can start that ceph-osd and things will recover.

Alternatively, if there is a catastrophic failure of `osd.1` (e.g., disk failure), we can tell the cluster that it is “lost” and to cope as best it can. Note that this is dangerous in that the cluster cannot guarantee that the other copies of the data are consistent and up to date. To instruct Ceph to continue anyway:

```
$ ceph osd lost 1
```

and recovery will proceed.

## Unfound objects

Under certain combinations of failures Ceph may complain about “unfound” objects:

```
$ ceph health detail
HEALTH_WARN 1 pgs degraded; 78/3778 unfound (2.065%)
pg 2.4 is active+degraded, 78 unfound
```

This means that the storage cluster knows that some objects (or newer copies of existing objects) exist, but it hasn’t found copies of them. One example of how this might come about for a PG whose data is on ceph-osds A and B:

- A goes down
- B handles some writes, alone
- A comes up
- A and B repeer, and the objects missing on A are queued for recovery.
- Before the new objects are copied, B goes down.

Now A knows that these object exist, but there is no live ceph-osd who has a copy. In this case, IO to those objects will block, and the cluster will hope that the failed node comes back soon; this is assumed to be preferable to returning an IO error to the user.

First, you can identify which objects are unfound with:

```
$ ceph pg 2.4 list_missing [starting offset, in json]
```

```
{ "offset": { "oid": "",
  "key": "",
  "snapid": 0,
```

```

    "hash": 0,
    "max": 0},
  "num_missing": 0,
  "num_unfound": 0,
  "objects": [
    { "oid": "object 1",
      "key": "",
      "hash": 0,
      "max": 0 },
    ...
  ],
  "more": 0}

```

If there are too many objects to list in a single result, the `more` field will be true and you can query for more. (Eventually the command line tool will hide this from you, but not yet.)

Second, you can identify which OSDs have been probed or might contain data:

```

$ ceph pg 2.4 query
...
  "recovery_state": [
    { "name": "Started/Primary/Active",
      "enter_time": "2012-03-06 15:15:46.713212",
      "might_have_unfound": [
        { "osd": 1,
          "status": "osd is down" } ] },

```

In this case, for example, the cluster knows that `osd.1` might have data, but it is down. The full range of possible states include:

```

* already probed
* querying
* osd is down
* not queried (yet)

```

Sometimes it simply takes some time for the cluster to query possible locations.

It is possible that there are other locations where the object can exist that are not listed. For example, if a `ceph-osd` is stopped and taken out of the cluster, the cluster fully recovers, and due to some future set of failures ends up with an unfound object, it won't consider the long-departed `ceph-osd` as a potential location to consider. (This scenario, however, is unlikely.)

If all possible locations have been queried and objects are still lost, you may have to give up on the lost objects. This, again, is possible given unusual combinations of failures that allow the cluster to learn about writes that were performed before the writes themselves are recovered. To mark the “unfound” objects as “lost”:

```

$ ceph pg 2.5 mark_unfound_lost revert

```

This the final argument specifies how the cluster should deal with lost objects. Currently the only supported option is “revert”, which will either roll back to a previous version of the object or (if it was a new object) forget about it entirely. Use this with caution, as it may confuse applications that expected the object to exist.

### Slow or unresponsive ceph-osd

If, for some reason, a `ceph-osd` is slow to respond to a request, it will generate log messages complaining about requests that are taking too long. The warning threshold defaults to 30 seconds, and is configurable via the `osd op complaint time` option. When this happens, the cluster log will receive messages like:

```
osd.0 192.168.106.220:6800/18813 312 : [WRN] old request osd_op(client.5099.0:790 fatty_26485_object)
```

Possible causes include:

- bad disk (check `dmesg` output)
- kernel file system bug (check `dmesg` output)
- overloaded cluster (check system load, `iostat`, etc.)
- ceph-osd bug

### Flapping OSDs

If something is causing OSDs to “flap” (repeated get marked down and then up again), you can force the monitors to stop with:

```
$ ceph osd set noup          # prevent osds from getting marked up
$ ceph osd set nodown       # prevent osds from getting marked down
```

These flags are recorded in the `osdmap` structure:

```
$ ceph osd dump | grep flags
flags no-up,no-down
```

You can clear the flags with:

```
$ ceph osd unset noup
$ ceph osd unset nodown
```

Two other flags are supported, `noin` and `noout`, which prevent booting OSDs from being marked in (allocated data) or down ceph-osds from eventually being marked out (regardless of what the current value for `mon osd down out interval` is).

Note that `noup`, `noout`, and `noout` are temporary in the sense that once the flags are cleared, the action they were blocking should occur shortly after. The `noin` flag, on the other hand, prevents ceph-osds from being marked in on boot, and any daemons that started while the flag was set will remain that way.

### Recovering from ceph-mds failure

### Recovering from radosgw failure

#### HTTP Request Errors

Examining the access and error logs for the web server itself is probably the first step in identifying what is going on. If there is a 500 error, that usually indicates a problem communicating with the `radosgw` daemon. Ensure the daemon is running, its socket path is configured, and that the web server is looking for it in the proper location.

#### Crashed radosgw process

If the `radosgw` process dies, you will normally see a 500 error from the web server (apache, nginx, etc.). In that situation, simply restarting `radosgw` will restore service.

To diagnose the cause of the crash, check the log in `/var/log/ceph` and/or the core file (if one was generated).

## Blocked radosgw Requests

If some (or all) radosgw requests appear to be blocked, you can get some insight into the internal state of the radosgw daemon via its admin socket. By default, there will be a socket configured to reside in `/var/run/ceph`, and the daemon can be queried with:

```
$ ceph --admin-daemon /var/run/ceph/client.rgw help
help                list available commands
objecter_requests   show in-progress osd requests
perfcounters_dump   dump perfcounters value
perfcounters_schema dump perfcounters schema
version             get protocol version
```

Of particular interest:

```
$ ceph --admin-daemon /var/run/ceph/client.rgw objecter_requests
...
```

will dump information about current in-progress requests with the RADOS cluster. This allows one to identify if any requests are blocked by a non-responsive ceph-osd. For example, one might see:

```
{ "ops": [
  { "tid": 1858,
    "pg": "2.d2041a48",
    "osd": 1,
    "last_sent": "2012-03-08 14:56:37.949872",
    "attempts": 1,
    "object_id": "fatty_25647_object1857",
    "object_locator": "@2",
    "snapid": "head",
    "snap_context": "0=[]",
    "mtime": "2012-03-08 14:56:37.949813",
    "osd_ops": [
      "write 0~4096"
    ]},
  { "tid": 1873,
    "pg": "2.695e9f8e",
    "osd": 1,
    "last_sent": "2012-03-08 14:56:37.970615",
    "attempts": 1,
    "object_id": "fatty_25647_object1872",
    "object_locator": "@2",
    "snapid": "head",
    "snap_context": "0=[]",
    "mtime": "2012-03-08 14:56:37.970555",
    "osd_ops": [
      "write 0~4096"
    ]},
  "linger_ops": [],
  "pool_ops": [],
  "pool_stat_ops": [],
  "statfs_ops": []}
```

In this dump, two requests are in progress. The `last_sent` field is the time the RADOS request was sent. If this is a while ago, it suggests that the OSD is not responding. For example, for request 1858, you could check the OSD status with:

```
$ ceph pg map 2.d2041a48
osdmap e9 pg 2.d2041a48 (2.0) -> up [1,0] acting [1,0]
```

This tells us to look at `osd.1`, the primary copy for this PG:

```
$ ceph --admin-daemon /var/run/ceph/osd.1.asok
{ "num_ops": 651,
  "ops": [
    { "description": "osd_op(client.4124.0:1858 fatty_25647_object1857 [write 0~4096] 2.d2041a48)",
      "received_at": "1331247573.344650",
      "age": "25.606449",
      "flag_point": "waiting for sub ops",
      "client_info": { "client": "client.4124",
                       "tid": 1858}},
    ...
  ]
}
```

The `flag_point` field indicates that the OSD is currently waiting for replicas to respond, in this case `osd.0`.

## 8.1.4 Adjusting the CRUSH map

There are a few ways to adjust the crush map:

- online, by issuing commands to the monitor
- offline, by extracting the current map to a file, modifying it, and then reinjecting a new map

For offline changes, some can be made directly with `crushtool`, and others require you to decompile the file to text form, manually edit it, and then recompile.

### Adding a new device (OSD) to the map

Adding new devices or moving existing devices to new positions in the CRUSH hierarchy can be done via the monitor. The general form is:

```
$ ceph osd crush set <id> <name> <weight> [<loc> [<lo2> ...]]
```

where

- `id` is the numeric device id (the OSD id)
- `name` is an alphanumeric name. By convention Ceph uses `osd.$id`.
- `weight` is a floating point weight value controlling how much data the device will be allocated. A decent convention is to make this the number of TB the device will store.
- `loc` is a list of `what=where` pairs indicating where in the CRUSH hierarchy the device will be stored. By default, the hierarchy (the `what`'s`) includes ``pool` (the default pool is normally the root of the hierarchy), `rack`, and `host`. At least one of these location specifiers has to refer to an existing point in the hierarchy, and only the lowest (most specific) match counts. Beneath that point, any intervening branches will be created as needed. Specifying the complete location is always sufficient, and also safe in that existing branches (and devices) won't be moved around.

For example, if the OSD id is 123, we want a weight of 1.0 and the device is on host `hostfoo` and rack `rackbar`:

```
$ ceph osd crush set 123 osd.123 1.0 pool=default rack=rackbar host=hostfoo
```

will add it to the hierarchy, or move it from its previous position. The rack `rackbar` and host `hostfoo` will be added as needed, as long as the pool `default` exists (as it does in the default Ceph CRUSH map generated during cluster creation).

Note that if I later add another device in the same host but specify a different pool or rack:

```
$ ceph osd crush set 124 osd.124 1.0 pool=nondefault rack=weirdrack host=hostfoo
```

the device will still be placed in host `hostfoo` at its current location (rack `rackbar` and pool `default`).

## Moving a bucket to a different position in the hierarchy

To move an existing bucket to a different position in the hierarchy, identify the bucket to move by name and specify the new location in the same fashion as with `osd crush set ...`:

```
$ ceph osd crush move <bucket name> [<loc> [<loc2> ...]]
```

where

- `name` is the name of the bucket to move. (To move a device, see *adjusting-crush-set*.)
- `loc` is a list of `what=where` pairs indicating where the bucket should be moved. See *adjusting-crush-set*.

## Adjusting the CRUSH weight

You can adjust the CRUSH weight for a device with:

```
$ ceph osd crush reweight osd.123 2.0
```

## Removing a device

You can remove a device from the crush map with:

```
$ ceph osd crush remove osd.123
```

## 8.1.5 Managing RADOS pools

### Creating new pools

### Authorizing access to pools

---

#### Todo

“Pool access is decided by having capability blah. To add the capability, blah blah, see *Setting capabilities for a key*”

---

#### Todo

when and who needs pool access

---

## Custom pool layouts with CRUSH

## 8.1.6 Managing Cephfs

### Mounting

### Kernel client

---

#### Todo

one time, fstab

---

---

## FUSE

---

### Todo

one time, fstab

---

## Using custom pools for subtrees

---

### Todo

- ./ceph usage
  - ./rados usage
- 

## 8.2 Radosgw installation and administration

RADOS Gateway (radosgw or rgw) provides a RESTful API to the object store. It interfaces with a web server via FastCGI, and with RADOS via libradospp.

### 8.2.1 Configuring Ceph for RADOS Gateway

In order for a host to act as a RADOS gateway, you must add a `[client.radosgw.<name>]` section to your Ceph configuration file (typically `/etc/ceph/ceph.conf`):

```
[client.radosgw.gateway]
  host = gateway
  rgw socket path = /tmp/radosgw.sock
```

`host` is the name of the host running radosgw. `keyring` points to the keyring file for Cephx authentication. `rgw socket path` is the location of the UNIX socket which radosgw binds to.

If your Ceph cluster has Cephx authentication enabled (highly recommended) you also need to add the following option to tell radosgw where it finds its authentication key:

```
[client.radosgw.gateway]
  keyring = /etc/ceph/keyring.radosgw.gateway
```

### 8.2.2 Creating authentication credentials

To allow radosgw to successfully authenticate with the Ceph cluster, use the `ceph-authtool` command to create a key and set its capabilities:

```
ceph-authtool -C -n client.radosgw.gateway \
  --gen-key /etc/ceph/keyring.radosgw.gateway
ceph-authtool -n client.radosgw.gateway \
  --cap mon 'allow r' --cap osd 'allow rwx' --cap mds 'allow' \
  /etc/ceph/keyring.radosgw.gateway
```



Finally, add this key to the authentication entries:

```
ceph auth add client.radosgw.gateway \
  --in-file=/etc/ceph/keyring.radosgw.gateway
```

## 8.2.3 Configuring the web server for radosgw

### 8.2.4 The radosgw FastCGI wrapper

A wrapper script, customarily named `radosgw.cgi` needs to go into your preferred location – typically your web server root directory.

```
#!/bin/sh
exec /usr/bin/radosgw -c /etc/ceph/ceph.conf -n client.radosgw.gateway
```

The `-c` option may be omitted if your Ceph configuration file resides in its default location (`/etc/ceph/ceph.conf`). The `-n` option identifies the client section in the configuration file that radosgw should parse – if omitted, this would default to `client.admin`.

### 8.2.5 Configuring Apache for radosgw

The recommended way of deploying radosgw is with Apache and `mod_fastcgi`. Ensure that both `mod_fastcgi` and `mod_rewrite` are enabled in your Apache configuration. Set the `FastCGIExternalServer` option to point to the radosgw FastCGI wrapper.

```
<IfModule mod_fastcgi.c>
  FastCgiExternalServer /var/www/radosgw.fcgi -socket /tmp/radosgw.sock
</IfModule>
```

Then, create a virtual host configuration as follows:

```
<VirtualHost *:80>
  ServerName radosgw.example.com
  ServerAlias rgw.example.com
  ServerAdmin webmaster@example.com
  DocumentRoot /var/www

  <IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteRule ^/(.*) /radosgw.fcgi?%{QUERY_STRING} [E=HTTP_AUTHORIZATION:%{HTTP:Authorization},L]
  </IfModule>

  <IfModule mod_fastcgi.c>
    <Directory /var/www>
      Options +ExecCGI
      AllowOverride All
      SetHandler fastcgi-script
      Order allow,deny
      Allow from all
      AuthBasicAuthoritative Off
    </Directory>
  </IfModule>

  AllowEncodedSlashes On
  ServerSignature Off
</VirtualHost>
```

## 8.2.6 Starting the daemons

For the gateway to become operational, start both the radosgw daemon and your web server:

```
service radosgw start
service apache start
```

## 8.2.7 Creating users

In order to be able to use the RESTful API, create a user with the `radosgw-admin` utility:

```
$ radosgw-admin user create --uid=johndoe --display-name="John Doe" --email=john@example.com
{ "user_id": "johndoe",
  "rados_uid": 0,
  "display_name": "John Doe",
  "email": "john@example.com",
  "suspended": 0,
  "subusers": [],
  "keys": [
    { "user": "johndoe",
      "access_key": "QFAMEDSJP5DEKJO0DDXY",
      "secret_key": "iaSFLDVvDdQt6lkNzHyW4fPLZugBAI1g17LO0+87"}],
  "swift_keys": []}
```

Note that creating a user also creates an `access_key` and `secret_key` entry for use with any S3 API-compatible client.

## 8.2.8 Enabling Swift access

Allowing access to the object store with Swift (OpenStack Object Storage) compatible clients requires an additional step, the creation of a subuser and a Swift access key.

```
# radosgw-admin subuser create --uid=johndoe --subuser=johndoe:swift --access=full
{ "user_id": "johndoe",
  "rados_uid": 0,
  "display_name": "John Doe",
  "email": "john@example.com",
  "suspended": 0,
  "subusers": [
    { "id": "johndoe:swift",
      "permissions": "full-control"}],
  "keys": [
    { "user": "johndoe",
      "access_key": "QFAMEDSJP5DEKJO0DDXY",
      "secret_key": "iaSFLDVvDdQt6lkNzHyW4fPLZugBAI1g17LO0+87"}],
  "swift_keys": []}

# radosgw-admin key create --subuser=johndoe:swift --key-type=swift
{ "user_id": "johndoe",
  "rados_uid": 0,
  "display_name": "John Doe",
  "email": "john@example.com",
  "suspended": 0,
  "subusers": [
    { "id": "johndoe:swift",
      "permissions": "full-control"}],
```

```
"keys": [
  { "user": "johndoe",
    "access_key": "QFAMEDSJP5DEKJO0DDXY",
    "secret_key": "iaSFLDVvDdQt6lkNzHyW4fPLZugBAI1g17LO0+87"}],
"swift_keys": [
  { "user": "johndoe:swift",
    "secret_key": "E9T2rUZNu2gxUjcwUBO8n\Ev4KX6\GprEuH4qhul"}] }
```

With this configuration, you are able to use any Swift client to connect to and use radosgw. As an example, you might use the `swift` command-line client utility that ships with the OpenStack Object Storage packages.

```
$ swift -V 1.0 -A http://radosgw.example.com/auth \
-U johndoe:swift -K E9T2rUZNu2gxUjcwUBO8n\Ev4KX6\GprEuH4qhul \
post test
$ swift -V 1.0 -A http://radosgw.example.com/auth \
-U johndoe:swift -K E9T2rUZNu2gxUjcwUBO8n\Ev4KX6\GprEuH4qhul \
upload test myfile
```

Note that the radosgw `user:subuser` tuple maps to the `tenant:user` tuple expected by Swift.

Note also that the radosgw Swift authentication service only supports built-in Swift authentication (`-V 1.0`) at this point. There is currently no way to make radosgw authenticate users via OpenStack Identity Service (Keystone).

## 8.3 RBD setup and administration

## 8.4 Monitoring Ceph

---

**Todo**

write me

---



# RECOMMENDATIONS

There are several ways to set up a storage system. Often, the nature of the load means different tradeoffs make sense for different installations. This section aims to walk through some common scenarios, and inform you on how to decide what is best for your cluster.

## 9.1 Hardware

See Hardware Recommendations for details.

## 9.2 Filesystem

For details on file systems when configuring a cluster See Hard Disk and File System Recommendations .

---

**Tip:** We recommend configuring Ceph to use the XFS file system in the near term, and `btrfs` in the long term once it is stable enough for production.

---

Before `ext3`, ReiserFS was the only journaling file system available for Linux. However, `ext3` doesn't provide Extended Attribute (XATTR) support. While `ext4` provides XATTR support, it only allows XATTRs up to 4kb. The 4kb limit is not enough for RADOS GW ACLs, snapshots, and other features. As of version 0.45, Ceph provides a `leveldb` feature for `ext4` file systems that stores XATTRs in excess of 4kb in a `leveldb` database.

The XFS and `btrfs` file systems provide numerous advantages in highly scaled data storage environments when compared to `ext3` and `ext4`. Both XFS and `btrfs` are [journaling file systems](#), which means that they are more robust when recovering from crashes, power outages, etc. These filesystems journal all of the changes they will make before performing writes.

XFS was developed for Silicon Graphics, and is a mature and stable filesystem. By contrast, `btrfs` is a relatively new file system that aims to address the long-standing wishes of system administrators working with large scale data storage environments. `btrfs` has some unique features and advantages compared to other Linux filesystems.

`btrfs` is a [copy-on-write](#) filesystem. It supports file creation timestamps and checksums that verify metadata integrity, so it can detect bad copies of data and fix them with the good copies. The copy-on-write capability means that `btrfs` can support snapshots that are writable. `btrfs` supports transparent compression and other features.

`btrfs` also incorporates multi-device management into the file system, which enables you to support heterogeneous disk storage infrastructure, data allocation policies. The community also aims to provide `fsck`, deduplication, and data encryption support in the future. This compelling list of features makes `btrfs` the ideal choice for Ceph clusters.

## 9.3 Data placement

---

### **Todo**

brief intro to CRUSH, pointers to more

---

### **Todo**

Considerations and tradeoffs of different placement policies

---

## 9.4 Disabling cryptography

Authentication is optional but very much recommended.

---

### **Todo**

write me

---

# CONTROL COMMANDS

## 10.1 Monitor commands

Monitor commands are issued using the ceph utility:

```
$ ceph [-m monhost] command
```

where the command is usually (though not always) of the form:

```
$ ceph subsystem command
```

## 10.2 System commands

```
$ ceph -s  
$ ceph status
```

Shows an overview of the current status of the cluster.

```
$ ceph -w
```

Shows a running summary of the status of the cluster, and major events.

```
$ ceph quorum_status
```

Show the monitor quorum, including which monitors are participating and which one is the leader.

```
$ ceph [-m monhost] mon_status
```

Query the status of a single monitor, including whether or not it is in the quorum.

## 10.3 AUTH subsystem

```
$ ceph auth add <osd> <--in-file|-i> <path-to-osd-keyring>
```

Add auth keyring for an osd.

```
$ ceph auth list
```

Show auth key OSD subsystem.

## 10.4 PG subsystem

```
$ ceph -- pg dump [--format <format>]
```

Output the stats of all pgs. Valid formats are “plain” and “json”, and plain is the default.

```
$ ceph -- pg dump_stuck inactive|unclean|stale [--format <format>] [-t|--threshold <seconds>]
```

Output the stats of all PGs stuck in the specified state.

--format may be plain (default) or json

--threshold defines how many seconds “stuck” is (default: 300)

**Inactive** PGs cannot process reads or writes because they are waiting for an OSD with the most up-to-date data to come back.

**Unclean** PGs contain objects that are not replicated the desired number of times. They should be recovering.

**Stale** PGs are in an unknown state - the OSDs that host them have not reported to the monitor cluster in a while (configured by mon\_osd\_report\_timeout).

```
$ ceph pg <pgid> mark_unfound_lost revert
```

Revert “lost” objects to their prior state, either a previous version or delete them if they were just created.

## 10.5 OSD subsystem

```
$ ceph osd stat
```

Query osd subsystem status.

```
$ ceph osd getmap -o file
```

Write a copy of the most recent osd map to a file. See [osdmaptool](#).

```
$ ceph osd getcrushmap -o file
```

Write a copy of the crush map from the most recent osd map to file. This is functionally equivalent to

```
$ ceph osd getmap -o /tmp/osdmap
$ osdmaptool /tmp/osdmap --export-crush file
```

```
$ ceph osd dump [--format <format>]
```

Dump the osd map. Valid formats for -f are “plain” and “json”. If no --format option is given, the osd map is dumped as plain text.

```
$ ceph osd tree [--format <format>]
```

Dump the osd map as a tree with one line per osd containing weight and state.

```
$ ceph osd crush set <id> <name> <weight> [<loc1> [<loc2> ...]]
```

Add or move a new item (osd) with the given id/name/weight at the specified location.

```
$ ceph osd crush remove <id>
```

Remove an existing item from the crush map.



```
$ ceph osd crush move <id> <loc1> [<loc2> ...]
```

Move an existing bucket from one position in the hierarchy to another.

```
$ ceph osd crush reweight <name> <weight>
```

Set the weight of the item given by <name> to <weight>.

```
$ ceph osd cluster_snap <name>
```

Create a cluster snapshot.

```
$ ceph osd lost [--yes-i-really-mean-it]
```

Mark an OSD as lost. This may result in permanent data loss. Use with caution.

```
$ ceph osd create [<id>]
```

Create a new OSD. If no ID is given, a new ID is automatically selected if possible.

```
$ ceph osd rm [<id>...]
```

Remove the given OSD(s).

```
$ ceph osd getmaxosd
```

Query the current max\_osd parameter in the osd map.

```
$ ceph osd setmap -i file
```

Import the given osd map. Note that this can be a bit dangerous, since the osd map includes dynamic state about which OSDs are current on or offline; only do this if you've just modified a (very) recent copy of the map.

```
$ ceph osd setcrushmap -i file
```

Import the given crush map.

```
$ ceph osd setmaxosd
```

Set the max\_osd parameter in the osd map. This is necessary when expanding the storage cluster.

```
$ ceph osd down N
```

Mark osdN down.

```
$ ceph osd out N
```

Mark osdN out of the distribution (i.e. allocated no data).

```
$ ceph osd in N
```

Mark osdN in the distribution (i.e. allocated data).

```
$ ceph class list
```

List classes that are loaded in the ceph cluster.

```
$ ceph osd pause
$ ceph osd unpause
```

Set or clear the pause flags in the OSD map. If set, no IO requests will be sent to any OSD. Clearing the flags via unpause results in resending pending requests.

```
$ ceph osd reweight N W
```

Set the weight of osdN to W. Two OSDs with the same weight will receive roughly the same number of I/O requests and store approximately the same amount of data.

```
$ ceph osd reweight-by-utilization [threshold]
```

Reweights all the OSDs by reducing the weight of OSDs which are heavily overused. By default it will adjust the weights downward on OSDs which have 120% of the average utilization, but if you include threshold it will use that percentage instead.

```
$ ceph osd blacklist add ADDRESS[:source_port] [TIME]
$ ceph osd blacklist rm ADDRESS[:source_port]
```

Adds/removes the address to/from the blacklist. When adding an address, you can specify how long it should be blacklisted in seconds; otherwise it will default to 1 hour. A blacklisted address is prevented from connecting to any osd. Blacklisting is most often used to prevent a laggy mds making bad changes to data on the osds.

These commands are mostly only useful for failure testing, as blacklists are normally maintained automatically and shouldn't need manual intervention.

```
$ ceph osd pool mksnap POOL SNAPNAME
$ ceph osd pool rmsnap POOL SNAPNAME
```

Creates/deletes a snapshot of a pool.

```
$ ceph osd pool create POOL [pg_num [pgp_num]]
$ ceph osd pool delete POOL
$ ceph osd pool rename OLDNAME NEWNAME
```

Creates/deletes/renames a storage pool.

```
$ ceph osd pool set POOL FIELD VALUE
```

Changes a pool setting. Valid fields are:

- `size`: Sets the number of copies of data in the pool.
- `crash_replay_interval`: The number of seconds to allow clients to replay acknowledged but uncommitted requests.
- `pg_num`: The placement group number.
- `pgp_num`: Effective number when calculating pg placement.
- `crush_ruleset`: rule number for mapping placement.

```
$ ceph osd pool get POOL FIELD
```

Get the value of a pool setting. Valid fields are:

- `pg_num`: See above.
- `pgp_num`: See above.
- `lpg_num`: The number of local PGs.
- `lpgp_num`: The number used for placing the local PGs.

```
$ ceph osd scrub N
```

Sends a scrub command to osdN. To send the command to all osds, use \*. TODO: what does this actually do

```
$ ceph osd repair N
```

Sends a repair command to osdN. To send the command to all osds, use \*. TODO: what does this actually do

```
$ ceph osd tell N bench [BYTES_PER_WRITE] [TOTAL_BYTES]
```

Runs a simple throughput benchmark against osdN, writing TOTAL\_BYTES in write requests of BYTES\_PER\_WRITE each. By default, the test writes 1 GB in total in 4-MB increments.

## 10.6 MDS subsystem

Change configuration parameters on a running mds.

```
$ ceph mds tell <mds-id> injectargs '--<switch> <value> [--<switch> <value>]'
```

Example:

```
$ ceph mds tell 0 injectargs '--debug_ms 1 --debug_mds 10'
```

Enables debug messages.

```
$ ceph mds stat
```

Displays the status of all metadata servers.

---

### Todo

ceph mds subcommands missing docs: set\_max\_mds, dump, getmap, stop, setmap

---

## 10.7 Mon subsystem

Show monitor stats:

```
$ ceph mon stat
2011-12-14 10:40:59.044395 mon <- [mon,stat]
2011-12-14 10:40:59.057111 mon.1 -> 'e3: 5 mons at {a=10.1.2.3:6789/0,b=10.1.2.4:6789/0,c=10.1.2.5:6789/0,d=10.1.2.6:6789/0,e=10.1.2.7:6789/0}
```

The quorum list at the end lists monitor nodes that are part of the current quorum.

This is also available more directly:

```
$ ./ceph quorum_status
2011-12-14 10:44:20.417705 mon <- [quorum_status]
2011-12-14 10:44:20.431890 mon.0 -> '{ "election_epoch": 10,
  "quorum": [
    0,
    1,
    2],
  "monmap": { "epoch": 1,
    "fsid": "444b489c-4f16-4b75-83f0-cb8097468898",
    "modified": "2011-12-12 13:28:27.505520",
    "created": "2011-12-12 13:28:27.505520",
    "mons": [
      { "rank": 0,
        "name": "a",
```

```
    "addr": "127.0.0.1:6789\0"},
  { "rank": 1,
    "name": "b",
    "addr": "127.0.0.1:6790\0"},
  { "rank": 2,
    "name": "c",
    "addr": "127.0.0.1:6791\0"}}}]}' (0)
```

The above will block until a quorum is reached.

For a status of just the monitor you connect to (use `-m HOST:PORT` to select):

```
$ ./ceph mon_status
2011-12-14 10:45:30.644414 mon <- [mon_status]
2011-12-14 10:45:30.644632 mon.0 -> '{ "name": "a",
  "rank": 0,
  "state": "leader",
  "election_epoch": 10,
  "quorum": [
    0,
    1,
    2],
  "outside_quorum": [],
  "monmap": { "epoch": 1,
    "fsid": "444b489c-4f16-4b75-83f0-cb8097468898",
    "modified": "2011-12-12 13:28:27.505520",
    "created": "2011-12-12 13:28:27.505520",
    "mons": [
      { "rank": 0,
        "name": "a",
        "addr": "127.0.0.1:6789\0"},
      { "rank": 1,
        "name": "b",
        "addr": "127.0.0.1:6790\0"},
      { "rank": 2,
        "name": "c",
        "addr": "127.0.0.1:6791\0"}}}]}' (0)
```

A dump of the monitor state:

```
$ ceph mon dump
2011-12-14 10:43:08.015333 mon <- [mon,dump]
2011-12-14 10:43:08.015567 mon.0 -> 'dumped monmap epoch 1' (0)
epoch 1
fsid 444b489c-4f16-4b75-83f0-cb8097468898
last_changed 2011-12-12 13:28:27.505520
created 2011-12-12 13:28:27.505520
0: 127.0.0.1:6789/0 mon.a
1: 127.0.0.1:6790/0 mon.b
2: 127.0.0.1:6791/0 mon.c
```

# API DOCUMENTATION

## 11.1 Librados (C)

*Librados* provides low-level access to the RADOS service. For an overview of RADOS, see *Architecture of Ceph*.

### 11.1.1 Example: connecting and writing an object

To use *Librados*, you instantiate a `rados_t` variable (a cluster handle) and call `rados_create()` with a pointer to it:

```
int err;
rados_t cluster;

err = rados_create(&cluster, NULL);
if (err < 0) {
    fprintf(stderr, "%s: cannot create a cluster handle: %s\n", argv[0], strerror(-err));
    exit(1);
}
```

Then you configure your `rados_t` to connect to your cluster, either by setting individual values (`rados_conf_set()`), using a configuration file (`rados_conf_read_file()`), using command line options (`rados_conf_parse_argv()`), or an environment variable (`rados_conf_parse_env()`):

```
err = rados_conf_read_file(cluster, "/path/to/myceph.conf");
if (err < 0) {
    fprintf(stderr, "%s: cannot read config file: %s\n", argv[0], strerror(-err));
    exit(1);
}
```

Once the cluster handle is configured, you can connect to the cluster with `rados_connect()`:

```
err = rados_connect(cluster);
if (err < 0) {
    fprintf(stderr, "%s: cannot connect to cluster: %s\n", argv[0], strerror(-err));
    exit(1);
}
```

Then you open an “IO context”, a `rados_ioctx_t`, with `rados_ioctx_create()`:

```
rados_ioctx_t io;
char *poolname = "mypool";

err = rados_ioctx_create(cluster, poolname, &io);
```

```
if (err < 0) {
    fprintf(stderr, "%s: cannot open rados pool %s: %s\n", argv[0], poolname, strerror(-err));
    rados_shutdown(cluster);
    exit(1);
}
```

Note that the pool you try to access must exist.

Then you can use the RADOS data manipulation functions, for example write into an object called `greeting` with `rados_write_full()`:

```
err = rados_write_full(io, "greeting", "hello", 5);
if (err < 0) {
    fprintf(stderr, "%s: cannot write pool %s: %s\n", argv[0], poolname, strerror(-err));
    rados_ioctx_destroy(io);
    rados_shutdown(cluster);
    exit(1);
}
```

In the end, you'll want to close your IO context and connection to RADOS with `rados_ioctx_destroy()` and `rados_shutdown()`:

```
rados_ioctx_destroy(io);
rados_shutdown(cluster);
```

### 11.1.2 Asynchronous IO

When doing lots of IO, you often don't need to wait for one operation to complete before starting the next one. *Librados* provides asynchronous versions of several operations:

- `rados_aio_write()`
- `rados_aio_append()`
- `rados_aio_write_full()`
- `rados_aio_read()`

For each operation, you must first create a `rados_completion_t` that represents what to do when the operation is safe or complete by calling `rados_aio_create_completion()`. If you don't need anything special to happen, you can pass `NULL`:

```
rados_completion_t comp;
err = rados_aio_create_completion(NULL, NULL, NULL, &comp);
if (err < 0) {
    fprintf(stderr, "%s: could not create aio completion: %s\n", argv[0], strerror(-err));
    rados_ioctx_destroy(io);
    rados_shutdown(cluster);
    exit(1);
}
```

Now you can call any of the aio operations, and wait for it to be in memory or on disk on all replicas:

```
err = rados_aio_write(io, "foo", comp, "bar", 3, 0);
if (err < 0) {
    fprintf(stderr, "%s: could not schedule aio write: %s\n", argv[0], strerror(-err));
    rados_aio_release(comp);
    rados_ioctx_destroy(io);
    rados_shutdown(cluster);
    exit(1);
}
```

```

}
rados_wait_for_complete(comp); // in memory
rados_wait_for_safe(comp); // on disk

```

Finally, we need to free the memory used by the completion with `rados_aio_release()`:

```
rados_aio_release(comp);
```

You can use the callbacks to tell your application when writes are durable, or when read buffers are full. For example, if you wanted to measure the latency of each operation when appending to several objects, you could schedule several writes and store the ack and commit time in the corresponding callback, then wait for all of them to complete using `rados_aio_flush()` before analyzing the latencies:

```

typedef struct {
    struct timeval start;
    struct timeval ack_end;
    struct timeval commit_end;
} req_duration;

void ack_callback(rados_completion_t comp, void *arg) {
    req_duration *dur = (req_duration *) arg;
    gettimeofday(&dur->ack_end, NULL);
}

void commit_callback(rados_completion_t comp, void *arg) {
    req_duration *dur = (req_duration *) arg;
    gettimeofday(&dur->commit_end, NULL);
}

int output_append_latency(rados_ioctx_t io, const char *data, size_t len, size_t num_writes) {
    req_duration times[num_writes];
    rados_completion_t comps[num_writes];
    for (size_t i = 0; i < num_writes; ++i) {
        gettimeofday(&times[i].start, NULL);
        int err = rados_aio_create_completion((void*) &times[i], ack_callback, commit_callback);
        if (err < 0) {
            fprintf(stderr, "Error creating rados completion: %s\n", strerror(-err));
            return err;
        }
        char obj_name[100];
        snprintf(obj_name, sizeof(obj_name), "foo%d", (unsigned long)i);
        err = rados_aio_append(io, obj_name, comps[i], data, len);
        if (err < 0) {
            fprintf(stderr, "Error from rados_aio_append: %s", strerror(-err));
            return err;
        }
    }
    // wait until all requests finish *and* the callbacks complete
    rados_aio_flush(io);
    // the latencies can now be analyzed
    printf("Request # | Ack latency (s) | Commit latency (s)\n");
    for (size_t i = 0; i < num_writes; ++i) {
        // don't forget to free the completions
        rados_aio_release(comps[i]);
        struct timeval ack_lat, commit_lat;
        timersub(&times[i].ack_end, &times[i].start, &ack_lat);
        timersub(&times[i].commit_end, &times[i].start, &commit_lat);
        printf("%9ld | %8ld.%06ld | %10ld.%06ld\n", (unsigned long) i, ack_lat.tv_sec, ack_lat.tv_usec,

```

```
    return 0;
}
```

Note that all the `rados_completion_t` must be freed with `rados_aio_release()` to avoid leaking memory.

### 11.1.3 API calls

#### Struct `rados_pool_stat_t`

struct `rados_pool_stat_t`  
Usage information for a pool.

##### Members

uint64\_t `num_bytes`  
space used in bytes

uint64\_t `num_kb`  
space used in KB

uint64\_t `num_objects`  
number of objects in the pool

uint64\_t `num_object_clones`  
number of clones of objects

uint64\_t `num_object_copies`  
`num_objects * num_replicas`

uint64\_t `num_objects_missing_on_primary`

uint64\_t `num_objects_unfound`  
number of objects found on no OSDs

uint64\_t `num_objects_degraded`  
number of objects replicated fewer times than they should be (but found on at least one OSD)

uint64\_t `num_rd`

uint64\_t `num_rd_kb`

uint64\_t `num_wr`

uint64\_t `num_wr_kb`

#### Struct `rados_cluster_stat_t`

struct `rados_cluster_stat_t`  
Cluster-wide usage information.

##### Members

uint64\_t `kb`

uint64\_t `kb_used`

uint64\_t `kb_avail`



uint64\_t num\_objects

## Defines

CEPH\_OSD\_TMAP\_HDR  
 CEPH\_OSD\_TMAP\_SET  
 CEPH\_OSD\_TMAP\_CREATE  
 CEPH\_OSD\_TMAP\_RM  
 LIBRADOS\_VER\_MAJOR  
 LIBRADOS\_VER\_MINOR  
 LIBRADOS\_VER\_EXTRA  
 LIBRADOS\_VERSION  
 LIBRADOS\_VERSION\_CODE  
 LIBRADOS\_SUPPORTS\_WATCH

## Types

### rados\_t

A handle for interacting with a RADOS cluster.

It encapsulates all RADOS client configuration, including username, key for authentication, logging, and debugging. Talking different clusters – or to the same cluster with different users – requires different cluster handles.

### rados\_config\_t

rados\_config\_t

A handle for the ceph configuration context for the rados\_t cluster instance. This can be used to share configuration context/state (e.g., logging configuration) between librados instance.

**Warning:** The config context does not have independent reference counting. As such, a rados\_config\_t handle retrieved from a given rados\_t is only valid as long as that rados\_t.

### rados\_ioctx\_t

An io context encapsulates a few settings for all I/O operations done on it:

- pool - set when the io context is created (see `rados_ioctx_create()`)
- snapshot context for writes (see `rados_ioctx_selfmanaged_snap_set_write_ctx()`)
- snapshot id to read from (see `rados_ioctx_snap_set_read()`)
- object locator for all single-object operations (see `rados_ioctx_locator_set_key()`)

**Warning:** changing any of these settings is not thread-safe - librados users must synchronize any of these changes on their own, or use separate io contexts for each thread

**rados\_list\_ctx\_t**

An iterator for listing the objects in a pool.

Used with `rados_objects_list_open()`, `rados_objects_list_next()`, and `rados_objects_list_close()`.

**rados\_snap\_t**

The id of a snapshot.

**rados\_xattrs\_iter\_t**

An iterator for listing extended attributes on an object.

Used with `rados_getxattrs()`, `rados_getxattrs_next()`, and `rados_getxattrs_end()`.

**rados\_completion\_t**

Represents the state of an asynchronous operation - it contains the return value once the operation completes, and can be used to block until the operation is complete or safe.

**rados\_callback\_t**

Callbacks for asynchronous operations take two parameters:

- `cb` the completion that has finished
- `arg` application defined data made available to the callback function

**rados\_watchcb\_t**

Callback activated when a notify is received on a watched object.

Parameters are:

- `opcode` undefined
- `ver` version of the watched object
- `arg` application-specific data

---

**Note:** BUG: `opcode` is an internal detail that shouldn't be exposed

---

## Functions

void **rados\_version** (int *\*major*, int *\*minor*, int *\*extra*)

Get the version of librados.

The version number is `major.minor.extra`. Note that this is unrelated to the Ceph version number.

TODO: define version semantics, i.e.:

- incrementing `major` is for backwards-incompatible changes
- incrementing `minor` is for backwards-compatible changes
- incrementing `extra` is for bug fixes

**Parameters**

- **major** – where to store the major version number
- **minor** – where to store the minor version number
- **extra** – where to store the extra version number

int **rados\_create** (rados\_t \*cluster, const char \*const id)

Create a handle for communicating with a RADOS cluster.

Ceph environment variables are read when this is called, so if \$CEPH\_ARGS specifies everything you need to connect, no further configuration is necessary.

#### Parameters

- **cluster** – where to store the handle
- **id** – the user to connect as (i.e. admin, not client.admin)

**Returns** 0 on success, negative error code on failure

int **rados\_create\_with\_context** (rados\_t \*cluster, rados\_config\_t cct)

Initialize a cluster handle from an existing configuration.

Share configuration state with another rados\_t instance.

#### Parameters

- **cluster** – where to store the handle
- **cct\_** – the existing configuration to use

**Returns** 0 on success, negative error code on failure

int **rados\_connect** (rados\_t cluster)

Connect to the cluster.

---

**Note:** BUG: Before calling this, calling a function that communicates with the cluster will crash.

---

#### Precondition

The cluster handle is configured with at least a monitor address. If cephx is enabled, a client name and secret must also be set.

---

#### Postcondition

If this succeeds, any function in librados may be used

---

#### Parameters

- **cluster** – The cluster to connect to.

**Returns** 0 on success, negative error code on failure

void **rados\_shutdown** (rados\_t cluster)

Disconnects from the cluster.

For clean up, this is only necessary after `rados_connect ()` has succeeded.

**Warning:** This does not guarantee any asynchronous writes have completed. To do that, you must call `rados_aio_flush()` on all open io contexts.

---

#### Postcondition

the cluster handle cannot be used again

---

#### Parameters

- **cluster** – the cluster to shutdown

int **rados\_conf\_read\_file** (rados\_t cluster, const char \*path)

Configure the cluster handle using a Ceph config file.

If path is NULL, the default locations are searched, and the first found is used. The locations are:

- \$CEPH\_CONF (environment variable)
  - /etc/ceph/ceph.conf
  - ~/.ceph/config
  - ceph.conf (in the current working directory)
- 

#### Precondition

`rados_connect ()` has not been called on the cluster handle

---

#### Parameters

- **cluster** – cluster handle to configure
- **path** – path to a Ceph configuration file

**Returns** 0 on success, negative error code on failure

int **rados\_conf\_parse\_argv** (rados\_t cluster, int argc, const char \*\*argv)

Configure the cluster handle with command line arguments.

argv can contain any common Ceph command line option, including any configuration parameter prefixed by ‘-’ and replacing spaces with dashes or underscores. For example, the following options are equivalent:

- --mon-host 10.0.0.1:6789
  - --mon\_host 10.0.0.1:6789
  - -m 10.0.0.1:6789
- 

#### Precondition

`rados_connect ()` has not been called on the cluster handle

---

#### Parameters

- **cluster** – cluster handle to configure
- **argc** – number of arguments in argv
- **argv** – arguments to parse

**Returns** 0 on success, negative error code on failure

int **rados\_conf\_parse\_env** (rados\_t *cluster*, const char \**var*)

Configure the cluster handle based on an environment variable.

The contents of the environment variable are parsed as if they were Ceph command line options. If *var* is NULL, the CEPH\_ARGS environment variable is used.

---

#### Precondition

`rados_connect()` has not been called on the cluster handle

---

**Note:** BUG: this is not threadsafe - it uses a static buffer

---

#### Parameters

- **cluster** – cluster handle to configure
- **var** – name of the environment variable to read

**Returns** 0 on success, negative error code on failure

int **rados\_conf\_set** (rados\_t *cluster*, const char \**option*, const char \**value*)

Set a configuration option.

---

#### Precondition

`rados_connect()` has not been called on the cluster handle

---

#### Parameters

- **cluster** – cluster handle to configure
- **option** – option to set
- **value** – value of the option

#### Returns

0 on success, negative error code on failure

-ENOENT when the option is not a Ceph configuration option

int **rados\_conf\_get** (rados\_t *cluster*, const char \**option*, char \**buf*, size\_t *len*)

Get the value of a configuration option.

#### Parameters

- **cluster** – configuration to read
- **option** – which option to read
- **buf** – where to write the configuration value
- **len** – the size of *buf* in bytes

#### Returns

0 on success, negative error code on failure

-ENAMETOOLONG if the buffer is too short to contain the requested value

int **rados\_cluster\_stat** (rados\_t *cluster*, struct rados\_cluster\_stat\_t *\*result*)

Read usage info about the cluster.

This tells you total space, space used, space available, and number of objects. These are not updated immediately when data is written, they are eventually consistent.

#### Parameters

- **cluster** – cluster to query
- **result** – where to store the results

**Returns** 0 on success, negative error code on failure

int **rados\_pool\_list** (rados\_t *cluster*, char *\*buf*, size\_t *len*)

List objects in a pool.

Gets a list of pool names as NULL-terminated strings. The pool names will be placed in the supplied buffer one after another. After the last pool name, there will be two 0 bytes in a row.

If len is too short to fit all the pool name entries we need, we will fill as much as we can.

#### Parameters

- **cluster** – cluster handle
- **buf** – output buffer
- **len** – output buffer length

**Returns** length of the buffer we would need to list all pools

**rados\_config\_t rados\_cct** (rados\_t *cluster*)

Get a configuration handle for a rados cluster handle.

This handle is valid only as long as the cluster handle is valid.

#### Parameters

- **cluster** – cluster handle

**Returns** config handle for this cluster

uint64\_t **rados\_get\_instance\_id** (rados\_t *cluster*)

Get a global id for current instance.

This id is a unique representation of current connection to the cluster

#### Parameters

- **cluster** – cluster handle

**Returns** instance global id

int **rados\_ioctx\_create** (rados\_t *cluster*, const char *\*pool\_name*, rados\_ioctx\_t *\*ioctx*)

Create an io context.

The io context allows you to perform operations within a particular pool. For more details see rados\_ioctx\_t.

#### Parameters

- **cluster** – which cluster the pool is in

- **pool\_name** – name of the pool
- **ioctx** – where to store the io context

**Returns** 0 on success, negative error code on failure

void **rados\_ioctx\_destroy**(*rados\_ioctx\_t io*)

The opposite of `rados_ioctx_create`.

This just tells librados that you no longer need to use the io context. It may not be freed immediately if there are pending asynchronous requests on it, but you should not use an io context again after calling this function on it.

**Warning:** This does not guarantee any asynchronous writes have completed. You must call `rados_aio_flush()` on the io context before destroying it to do that.

#### Parameters

- **io** – the io context to dispose of

*rados\_config\_t* **rados\_ioctx\_cct**(*rados\_ioctx\_t io*)

Get configuration handle for a pool handle.

#### Parameters

- **io** – pool handle

**Returns** *rados\_config\_t* for this cluster

int **rados\_ioctx\_pool\_stat**(*rados\_ioctx\_t io*, struct *rados\_pool\_stat\_t \*stats*)

Get pool usage statistics.

Fills in a *rados\_pool\_stat\_t* after querying the cluster.

#### Parameters

- **io** – determines which pool to query
- **stats** – where to store the results

**Returns** 0 on success, negative error code on failure

int64\_t **rados\_pool\_lookup**(*rados\_t cluster*, const char \**pool\_name*)

Get the id of a pool.

#### Parameters

- **cluster** – which cluster the pool is in
- **pool\_name** – which pool to look up

#### Returns

id of the pool

-ENOENT if the pool is not found

int **rados\_pool\_create**(*rados\_t cluster*, const char \**pool\_name*)

Create a pool with default settings.

The default owner is the admin user (auid 0). The default crush rule is rule 0.

**Parameters**

- **cluster** – the cluster in which the pool will be created
- **pool\_name** – the name of the new pool

**Returns** 0 on success, negative error code on failure

```
int rados_pool_create_with_auid(rados_t cluster, const char *pool_name,
                               uint64_t auid)
```

Create a pool owned by a specific auid.

The auid is the authenticated user id to give ownership of the pool. TODO: document auid and the rest of the auth system

**Parameters**

- **cluster** – the cluster in which the pool will be created
- **pool\_name** – the name of the new pool
- **auid** – the id of the owner of the new pool

**Returns** 0 on success, negative error code on failure

```
int rados_pool_create_with_crush_rule(rados_t cluster, const char *pool_name,
                                     __u8 crush_rule_num)
```

Create a pool with a specific CRUSH rule.

**Parameters**

- **cluster** – the cluster in which the pool will be created
- **pool\_name** – the name of the new pool
- **crush\_rule\_num** – which rule to use for placement in the new pool1

**Returns** 0 on success, negative error code on failure

```
int rados_pool_create_with_all(rados_t cluster, const char *pool_name, uint64_t auid,
                              __u8 crush_rule_num)
```

Create a pool with a specific CRUSH rule and auid.

This is a combination of `rados_pool_create_with_crush_rule()` and `rados_pool_create_with_auid()`

**Parameters**

- **cluster** – the cluster in which the pool will be created
- **pool\_name** – the name of the new pool
- **crush\_rule\_num** – which rule to use for placement in the new pool2
- **auid** – the id of the owner of the new pool

**Returns** 0 on success, negative error code on failure

```
int rados_pool_delete(rados_t cluster, const char *pool_name)
```

Delete a pool and all data inside it.

The pool is removed from the cluster immediately, but the actual data is deleted in the background.

**Parameters**

- **cluster** – the cluster the pool is in



- **pool\_name** – which pool to delete

**Returns** 0 on success, negative error code on failure

int **rados\_ioctx\_pool\_set\_auid**(rados\_ioctx\_t *io*, uint64\_t *auid*)

Attempt to change an io context's associated auid "owner".

Requires that you have write permission on both the current and new auid.

#### Parameters

- **io** – reference to the pool to change.
- **auid** – the auid you wish the io to have.

**Returns** 0 on success, negative error code on failure

int **rados\_ioctx\_pool\_get\_auid**(rados\_ioctx\_t *io*, uint64\_t \**auid*)

Get the auid of a pool.

#### Parameters

- **io** – pool to query
- **auid** – where to store the auid

**Returns** 0 on success, negative error code on failure

int64\_t **rados\_ioctx\_get\_id**(rados\_ioctx\_t *io*)

Get the pool id of the io context.

#### Parameters

- **io** – the io context to query

**Returns** the id of the pool the io context uses

int **rados\_ioctx\_get\_pool\_name**(rados\_ioctx\_t *io*, char \**buf*, unsigned *maxlen*)

Get the pool name of the io context.

#### Parameters

- **io** – the io context to query
- **buf** – pointer to buffer where name will be stored
- **maxlen** – size of buffer where name will be stored

**Returns** length of string stored, or -ERANGE if buffer too small

void **rados\_ioctx\_locator\_set\_key**(rados\_ioctx\_t *io*, const char \**key*)

Set the key for mapping objects to pgs within an io context.

The key is used instead of the object name to determine which placement groups an object is put in. This affects all subsequent operations of the io context - until a different locator key is set, all objects in this io context will be placed in the same pg.

This is useful if you need to do clone\_range operations, which must be done with the source and destination objects in the same pg.

#### Parameters

- **io** – the io context to change
- **key** – the key to use as the object locator, or NULL to discard any previously set key

int **rados\_objects\_list\_open** (rados\_ioctx\_t *io*, rados\_list\_ctx\_t \**ctx*)  
Start listing objects in a pool.

#### Parameters

- **io** – the pool to list from
- **ctx** – the handle to store list context in

**Returns** 0 on success, negative error code on failure

int **rados\_objects\_list\_next** (rados\_list\_ctx\_t *ctx*, const char \*\**entry*, const char \*\**key*)  
Get the next object name and locator in the pool.

\**entry* and \**key* are valid until next call to rados\_objects\_list\_\*

#### Parameters

- **ctx** – iterator marking where you are in the listing
- **entry** – where to store the name of the entry
- **key** – where to store the object locator (set to NULL to ignore)

#### Returns

0 on success, negative error code on failure  
-ENOENT when there are no more objects to list

void **rados\_objects\_list\_close** (rados\_list\_ctx\_t *ctx*)  
Close the object listing handle.

This should be called when the handle is no longer needed. The handle should not be used after it has been closed.

#### Parameters

- **ctx** – the handle to close

int **rados\_ioctx\_snap\_create** (rados\_ioctx\_t *io*, const char \**snapname*)  
Create a pool-wide snapshot.

#### Parameters

- **io** – the pool to snapshot
- **snapname** – the name of the snapshot

**Returns** 0 on success, negative error code on failure

int **rados\_ioctx\_snap\_remove** (rados\_ioctx\_t *io*, const char \**snapname*)  
Delete a pool snapshot.

#### Parameters

- **io** – the pool to delete the snapshot from
- **snapname** – which snapshot to delete

**Returns** 0 on success, negative error code on failure

int **rados\_rollback** (rados\_ioctx\_t *io*, const char \**oid*, const char \**snapname*)

Rollback an object to a pool snapshot.

The contents of the object will be the same as when the snapshot was taken.

#### Parameters

- **io** – the pool in which the object is stored
- **oid** – the name of the object to rollback
- **snapname** – which snapshot to rollback to

**Returns** 0 on success, negative error code on failure

void **rados\_ioctx\_snap\_set\_read** (rados\_ioctx\_t *io*, rados\_snap\_t *snap*)

Set the snapshot from which reads are performed.

Subsequent reads will return data as it was at the time of that snapshot.

#### Parameters

- **io** – the io context to change
- **snap** – the id of the snapshot to set, or CEPH\_NOSNAP for no snapshot (i.e. normal operation)

int **rados\_ioctx\_selfmanaged\_snap\_create** (rados\_ioctx\_t *io*, rados\_snap\_t \**snapid*)

Allocate an ID for a self-managed snapshot.

Get a unique ID to put in the snapshot context to create a snapshot. A clone of an object is not created until a write with the new snapshot context is completed.

#### Parameters

- **io** – the pool in which the snapshot will exist
- **snapid** – where to store the newly allocated snapshot ID

**Returns** 0 on success, negative error code on failure

int **rados\_ioctx\_selfmanaged\_snap\_remove** (rados\_ioctx\_t *io*, rados\_snap\_t *snapid*)

Remove a self-managed snapshot.

This increases the snapshot sequence number, which will cause snapshots to be removed lazily.

#### Parameters

- **io** – the pool in which the snapshot will exist
- **snapid** – where to store the newly allocated snapshot ID

**Returns** 0 on success, negative error code on failure

int **rados\_ioctx\_selfmanaged\_snap\_rollback** (rados\_ioctx\_t *io*, const char \**oid*, rados\_snap\_t *snapid*)

Rollback an object to a self-managed snapshot.

The contents of the object will be the same as when the snapshot was taken.

#### Parameters

- **io** – the pool in which the object is stored
- **oid** – the name of the object to rollback
- **snapid** – which snapshot to rollback to

**Returns** 0 on success, negative error code on failure

```
int rados_ioctx_selfmanaged_snap_set_write_ctx (rados_ioctx_t io, rados_snap_t seq, rados_snap_t *snaps, int num_snaps)
```

Set the snapshot context for use when writing to objects.

This is stored in the io context, and applies to all future writes.

#### Parameters

- **io** – the io context to change
- **seq** – the newest snapshot sequence number for the pool
- **snaps** – array of snapshots in sorted by descending id
- **num\_snaps** – how many snapshots are in the snaps array

#### Returns

0 on success, negative error code on failure  
-EINVAL if snaps are not in descending order

```
int rados_ioctx_snap_list (rados_ioctx_t io, rados_snap_t *snaps, int maxlen)
```

List all the ids of pool snapshots.

If the output array does not have enough space to fit all the snapshots, -ERANGE is returned and the caller should retry with a larger array.

#### Parameters

- **io** – the pool to read from
- **snaps** – where to store the results
- **maxlen** – the number of rados\_snap\_t that fit in the snaps array

#### Returns

number of snapshots on success, negative error code on failure  
-ERANGE is returned if the snaps array is too short

```
int rados_ioctx_snap_lookup (rados_ioctx_t io, const char *name, rados_snap_t *id)
```

Get the id of a pool snapshot.

#### Parameters

- **io** – the pool to read from
- **name** – the snapshot to find
- **id** – where to store the result

**Returns** 0 on success, negative error code on failure

```
int rados_ioctx_snap_get_name(rados_ioctx_t io, rados_snap_t id, char *name,
                              int maxlen)
```

Get the name of a pool snapshot.

#### Parameters

- **io** – the pool to read from
- **id** – the snapshot to find
- **name** – where to store the result
- **maxlen** – the size of the name array

#### Returns

0 on success, negative error code on failure  
-ERANGE if the name array is too small

```
int rados_ioctx_snap_get_stamp(rados_ioctx_t io, rados_snap_t id, time_t *t)
```

Find when a pool snapshot occurred.

#### Parameters

- **io** – the pool the snapshot was taken in
- **id** – the snapshot to lookup
- **t** – where to store the result

**Returns** 0 on success, negative error code on failure

```
uint64_t rados_get_last_version(rados_ioctx_t io)
```

Return the version of the last object read or written to.

This exposes the internal version number of the last object read or written via this io context

#### Parameters

- **io** – the io context to check

**Returns** last read or written object version

```
int rados_write(rados_ioctx_t io, const char *oid, const char *buf, size_t len, uint64_t off)
```

Write data to an object.

#### Parameters

- **io** – the io context in which the write will occur
- **oid** – name of the object
- **buf** – data to write
- **len** – length of the data, in bytes
- **off** – byte offset in the object to begin writing at

**Returns** number of bytes written on success, negative error code on failure

int **rados\_write\_full** (rados\_ioctx\_t *io*, const char \**oid*, const char \**buf*, size\_t *len*)

Write an entire object.

The object is filled with the provided data. If the object exists, it is atomically truncated and then written.

#### Parameters

- **io** – the io context in which the write will occur
- **oid** – name of the object
- **buf** – data to write
- **len** – length of the data, in bytes

**Returns** 0 on success, negative error code on failure

int **rados\_clone\_range** (rados\_ioctx\_t *io*, const char \**dst*, uint64\_t *dst\_off*, const char \**src*,  
uint64\_t *src\_off*, size\_t *len*)

Efficiently copy a portion of one object to another.

If the underlying filesystem on the OSD supports it, this will be a copy-on-write clone.

The src and dest objects must be in the same pg. To ensure this, the io context should have a locator key set (see [rados\\_ioctx\\_locator\\_set\\_key\(\)](#)).

#### Parameters

- **io** – the context in which the data is cloned
- **dst** – the name of the destination object
- **dst\_off** – the offset within the destination object (in bytes)
- **src** – the name of the source object
- **src\_off** – the offset within the source object (in bytes)
- **len** – how much data to copy

**Returns** 0 on success, negative error code on failure

int **rados\_append** (rados\_ioctx\_t *io*, const char \**oid*, const char \**buf*, size\_t *len*)

Append data to an object.

#### Parameters

- **io** – the context to operate in
- **oid** – the name of the object
- **buf** – the data to append
- **len** – length of buf (in bytes)

**Returns** number of bytes written on success, negative error code on failure

int **rados\_read** (rados\_ioctx\_t *io*, const char \**oid*, char \**buf*, size\_t *len*, uint64\_t *off*)

Read data from an object.

The io context determines the snapshot to read from, if any was set by [rados\\_ioctx\\_snap\\_set\\_read\(\)](#).

#### Parameters

- **io** – the context in which to perform the read
- **oid** – the name of the object to read from
- **buf** – where to store the results
- **len** – the number of bytes to read
- **off** – the offset to start reading from in the object

**Returns** number of bytes read on success, negative error code on failure

int **rados\_remove** (rados\_ioctx\_t *io*, const char \**oid*)  
Delete an object.

---

**Note:** This does not delete any snapshots of the object.

---

#### Parameters

- **io** – the pool to delete the object from
- **oid** – the name of the object to delete

**Returns** 0 on success, negative error code on failure

int **rados\_trunc** (rados\_ioctx\_t *io*, const char \**oid*, uint64\_t *size*)  
Resize an object.

If this enlarges the object, the new area is logically filled with zeroes. If this shrinks the object, the excess data is removed.

#### Parameters

- **io** – the context in which to truncate
- **oid** – the name of the object
- **size** – the new size of the object in bytes

**Returns** 0 on success, negative error code on failure

int **rados\_getxattr** (rados\_ioctx\_t *io*, const char \**o*, const char \**name*, char \**buf*, size\_t *len*)  
Get the value of an extended attribute on an object.

#### Parameters

- **io** – the context in which the attribute is read
- **o** – name of the object
- **name** – which extended attribute to read
- **buf** – where to store the result
- **len** – size of buf in bytes

**Returns** length of xattr value on success, negative error code on failure

int **rados\_setxattr** (rados\_ioctx\_t *io*, const char \**o*, const char \**name*, const char \**buf*,  
size\_t *len*)  
Set an extended attribute on an object.

**Parameters**

- **io** – the context in which xattr is set
- **o** – name of the object
- **name** – which extended attribute to set
- **buf** – what to store in the xattr
- **len** – the number of bytes in buf

**Returns** 0 on success, negative error code on failure

int **rados\_rmattr**(rados\_ioctx\_t *io*, const char \**o*, const char \**name*)

Delete an extended attribute from an object.

**Parameters**

- **io** – the context in which to delete the xattr
- **o** – the name of the object
- **name** – which xattr to delete

**Returns** 0 on success, negative error code on failure

int **rados\_getxattrs**(rados\_ioctx\_t *io*, const char \**oid*, rados\_xattrs\_iter\_t \**iter*)

Start iterating over xattrs on an object.

---

**Postcondition**

iter is a valid iterator

---

**Parameters**

- **io** – the context in which to list xattrs
- **oid** – name of the object
- **iter** – where to store the iterator

**Returns** 0 on success, negative error code on failure

int **rados\_getxattrs\_next**(rados\_xattrs\_iter\_t *iter*, const char \*\**name*, const char \*\**val*,  
size\_t \**len*)

Get the next xattr on the object.

---

**Precondition**

iter is a valid iterator

---

---

**Postcondition**

name is the NULL-terminated name of the next xattr, and val contains the value of the xattr, which is of length len. If the end of the list has been reached, name and val are NULL, and len is 0.

---

**Parameters**

- **iter** – iterator to advance



- **name** – where to store the name of the next xattr
- **val** – where to store the value of the next xattr
- **len** – the number of bytes in val

**Returns** 0 on success, negative error code on failure

void **rados\_getxattrs\_end**([rados\\_xattrs\\_iter\\_t](#) iter)

Close the xattr iterator.

iter should not be used after this is called.

#### Parameters

- **iter** – the iterator to close

int **rados\_stat**([rados\\_ioctx\\_t](#) io, const char \*o, uint64\_t \*psize, time\_t \*pmtime)

Get object stats (size/mtime)

TODO: when are these set, and by whom? can they be out of date?

#### Parameters

- **io** – ioctx
- **o** – object name
- **psize** – where to store object size
- **pmtime** – where to store modification time

**Returns** 0 on success, negative error code on failure

int **rados\_tmap\_update**([rados\\_ioctx\\_t](#) io, const char \*o, const char \*cmdbuf, size\_t cmdbuflen)

Update tmap (trivial map)

Do compound update to a tmap object, inserting or deleting some number of records. cmdbuf is a series of operation byte codes, following by command payload. Each command is a single-byte command code, whose value is one of CEPH\_OSD\_TMAP\_\*.

#### •update tmap ‘header’

–1 byte = CEPH\_OSD\_TMAP\_HDR

–4 bytes = data length (little endian)

–N bytes = data

#### •insert/update one key/value pair

–1 byte = CEPH\_OSD\_TMAP\_SET

–4 bytes = key name length (little endian)

–N bytes = key name

–4 bytes = data length (little endian)

–M bytes = data

#### •insert one key/value pair; return -EEXIST if it already exists.

–1 byte = CEPH\_OSD\_TMAP\_CREATE

-4 bytes = key name length (little endian)

-N bytes = key name

-4 bytes = data length (little endian)

-M bytes = data

- remove one key/value pair

-1 byte = CEPH\_OSD\_TMAP\_RM

-4 bytes = key name length (little endian)

-N bytes = key name

Restrictions:

- The HDR update must precede any key/value updates.
- All key/value updates must be in lexicographically sorted order in `cmdbuf`.
- You can read/write to a tmap object via the regular APIs, but you should be careful not to corrupt it. Also be aware that the object format may change without notice.

#### Parameters

- **io** – iocx
- **o** – object name
- **cmdbuf** – command buffer
- **cmdbuflen** – command buffer length in bytes

**Returns** 0 on success, negative error code on failure

int **rados\_tmap\_put** ([rados\\_iocx\\_t](#) io, const char \*o, const char \*buf, size\_t buflen)

Store complete tmap (trivial map) object.

Put a full tmap object into the store, replacing what was there.

The format of buf is:

- 4 bytes - length of header (little endian)
- N bytes - header data
- 4 bytes - number of keys (little endian)

and for each key,

- 4 bytes - key name length (little endian)
- N bytes - key name
- 4 bytes - value length (little endian)
- M bytes - value data

#### Parameters

- **io** – iocx
- **o** – object name
- **buf** – buffer

- **buflen** – buffer length in bytes

**Returns** 0 on success, negative error code on failure

int **rados\_tmap\_get** (rados\_ioctx\_t *io*, const char \**o*, char \**buf*, size\_t *buflen*)

Fetch complete tmap (trivial map) object.

Read a full tmap object. See `rados_tmap_put()` for the format the data is returned in.

#### Parameters

- **io** – ioctx
- **o** – object name
- **buf** – buffer
- **buflen** – buffer length in bytes

#### Returns

0 on success, negative error code on failure

-ERANGE if buf isn't big enough

int **rados\_exec** (rados\_ioctx\_t *io*, const char \**oid*, const char \**cls*, const char \**method*, const char \**in\_buf*, size\_t *in\_len*, char \**buf*, size\_t *out\_len*)

Execute an OSDclass method on an object.

The OSD has a plugin mechanism for performing complicated operations on an object atomically. These plugins are called classes. This function allows librados users to call the custom methods. The input and output formats are defined by the class. Classes in ceph.git can be found in `src/cls_*.cc`

#### Parameters

- **io** – the context in which to call the method
- **oid** – the object to call the method on
- **cls** – the name of the class
- **method** – the name of the method
- **in\_buf** – where to find input
- **in\_len** – length of in\_buf in bytes
- **buf** – where to store output
- **out\_len** – length of buf in bytes

**Returns** the length of the output, or -ERANGE if out\_buf does not have enough space to store it (For methods that return data). For methods that don't return data, the return value is method-specific.

int **rados\_aio\_create\_completion** (void \**cb\_arg*, rados\_callback\_t *cb\_complete*, rados\_callback\_t *cb\_safe*, rados\_completion\_t \**pc*)

Constructs a completion to use with asynchronous operations.

The complete and safe callbacks correspond to operations being acked and committed, respectively. The callbacks are called in order of receipt, so the safe callback may be triggered before the complete callback, and vice versa. This is affected by journaling on the OSDs.

TODO: more complete documentation of this elsewhere (in the RADOS docs?)

**Note:** Read operations only get a complete callback.

BUG: this should check for ENOMEM instead of throwing an exception

---

#### Parameters

- **cb\_arg** – application-defined data passed to the callback functions
- **cb\_complete** – the function to be called when the operation is in memory on all replicas
- **cb\_safe** – the function to be called when the operation is on stable storage on all replicas
- **pc** – where to store the completion

**Returns** 0

int **rados\_aio\_wait\_for\_complete** (rados\_completion\_t c)

Block until an operation completes.

This means it is in memory on all replicas.

---

**Note:** BUG: this should be void

---

#### Parameters

- **c** – operation to wait for

**Returns** 0

int **rados\_aio\_wait\_for\_safe** (rados\_completion\_t c)

Block until an operation is safe.

This means it is on stable storage on all replicas.

---

**Note:** BUG: this should be void

---

#### Parameters

- **c** – operation to wait for

**Returns** 0

int **rados\_aio\_is\_complete** (rados\_completion\_t c)

Has an asynchronous operation completed?

|   |
|---|
| <b>Warning:</b> This does not imply that the complete callback has finished |
|---|

#### Parameters

- **c** – async operation to inspect

**Returns** whether c is complete

int **rados\_aio\_is\_safe** (rados\_completion\_t c)

Is an asynchronous operation safe?

**Warning:** This does not imply that the safe callback has finished

#### Parameters

- **c** – async operation to inspect

**Returns** whether c is safe

int **rados\_aio\_wait\_for\_complete\_and\_cb** (rados\_completion\_t c)

Block until an operation completes and callback completes.

This means it is in memory on all replicas and can be read.

---

**Note:** BUG: this should be void

---

#### Parameters

- **c** – operation to wait for

**Returns** 0

int **rados\_aio\_wait\_for\_safe\_and\_cb** (rados\_completion\_t c)

Block until an operation is safe and callback has completed.

This means it is on stable storage on all replicas.

---

**Note:** BUG: this should be void

---

#### Parameters

- **c** – operation to wait for

**Returns** 0

int **rados\_aio\_is\_complete\_and\_cb** (rados\_completion\_t c)

Has an asynchronous operation and callback completed.

#### Parameters

- **c** – async operation to inspect

**Returns** whether c is complete

int **rados\_aio\_is\_safe\_and\_cb** (rados\_completion\_t c)

Is an asynchronous operation safe and has the callback completed.

#### Parameters

- **c** – async operation to inspect

**Returns** whether c is safe

int **rados\_aio\_get\_return\_value** (rados\_completion\_t c)

Get the return value of an asynchronous operation.

The return value is set when the operation is complete or safe, whichever comes first.

---

#### Precondition

The operation is safe or complete

---

**Note:** BUG: complete callback may never be called when the safe message is received before the complete message

---

#### Parameters

- **c** – async operation to inspect

**Returns** return value of the operation

void **rados\_aio\_release** (rados\_completion\_t c)

Release a completion.

Call this when you no longer need the completion. It may not be freed immediately if the operation is not acked and committed.

#### Parameters

- **c** – completion to release

int **rados\_aio\_write** (rados\_ioctx\_t io, const char \*oid, rados\_completion\_t completion, const char \*buf, size\_t len, uint64\_t off)

Write data to an object asynchronously.

Queues the write and returns. The return value of the completion will be 0 on success, negative error code on failure.

#### Parameters

- **io** – the context in which the write will occur
- **oid** – name of the object
- **completion** – what to do when the write is safe and complete
- **buf** – data to write
- **len** – length of the data, in bytes
- **off** – byte offset in the object to begin writing at

**Returns** 0 on success, -EROFS if the io context specifies a snap\_seq other than CEPH\_NOSNAP

int **rados\_aio\_append** (rados\_ioctx\_t io, const char \*oid, rados\_completion\_t completion, const char \*buf, size\_t len)

Asynchronously append data to an object.

Queues the append and returns.

The return value of the completion will be 0 on success, negative error code on failure.

**Parameters**

- **io** – the context to operate in
- **oid** – the name of the object
- **completion** – what to do when the append is safe and complete
- **buf** – the data to append
- **len** – length of buf (in bytes)

**Returns** 0 on success, -EROFS if the io context specifies a snap\_seq other than CEPH\_NOSNAP

```
int rados_aio_write_full(rados_ioctx_t io, const char *oid, rados_completion_t completion,
                        const char *buf, size_t len)
```

Asynchronously write an entire object.

The object is filled with the provided data. If the object exists, it is atomically truncated and then written. Queues the write\_full and returns.

The return value of the completion will be 0 on success, negative error code on failure.

**Parameters**

- **io** – the io context in which the write will occur
- **oid** – name of the object
- **completion** – what to do when the write\_full is safe and complete
- **buf** – data to write
- **len** – length of the data, in bytes

**Returns** 0 on success, -EROFS if the io context specifies a snap\_seq other than CEPH\_NOSNAP

```
int rados_aio_read(rados_ioctx_t io, const char *oid, rados_completion_t completion,
                  char *buf, size_t len, uint64_t off)
```

Asynchronously read data from an object.

The io context determines the snapshot to read from, if any was set by `rados_ioctx_snap_set_read()`.

The return value of the completion will be number of bytes read on success, negative error code on failure.

---

**Note:** only the ‘complete’ callback of the completion will be called.

---

**Parameters**

- **io** – the context in which to perform the read
- **oid** – the name of the object to read from
- **completion** – what to do when the read is complete
- **buf** – where to store the results
- **len** – the number of bytes to read
- **off** – the offset to start reading from in the object

**Returns** 0 on success, negative error code on failure

int **rados\_aio\_flush**(rados\_ioctx\_t io)

Block until all pending writes in an io context are safe.

This is not equivalent to calling `rados_aio_wait_for_safe()` on all write completions, since this waits for the associated callbacks to complete as well.

---

**Note:** BUG: always returns 0, should be void or accept a timeout

---

#### Parameters

- **io** – the context to flush

**Returns** 0 on success, negative error code on failure

int **rados\_watch**(rados\_ioctx\_t io, const char \*o, uint64\_t ver, uint64\_t \*handle, rados\_watchcb\_t watchcb, void \*arg)

Register an interest in an object.

A watch operation registers the client as being interested in notifications on an object. OSDs keep track of watches on persistent storage, so they are preserved across cluster changes by the normal recovery process. If the client loses its connection to the primary OSD for a watched object, the watch will be removed after 30 seconds. Watches are automatically reestablished when a new connection is made, or a placement group switches OSDs.

---

**Note:** BUG: watch timeout should be configurable

BUG: librados should provide a way for watchers to notice connection resets

---

#### Parameters

- **io** – the pool the object is in
- **o** – the object to watch
- **ver** – expected version of the object
- **handle** – where to store the internal id assigned to this watch
- **watchcb** – what to do when a notify is received on this object
- **arg** – application defined data to pass when watchcb is called

#### Returns

0 on success, negative error code on failure

-ERANGE if the version of the object is greater than ver

int **rados\_unwatch**(rados\_ioctx\_t io, const char \*o, uint64\_t handle)

Unregister an interest in an object.

Once this completes, no more notifies will be sent to us for this watch. This should be called to clean up unneeded watchers.

#### Parameters

- **io** – the pool the object is in



- **o** – the name of the watched object
- **handle** – which watch to unregister

**Returns** 0 on success, negative error code on failure

int **rados\_notify** (rados\_ioctx\_t *io*, const char \**o*, uint64\_t *ver*, const char \**buf*, int *buf\_len*)

Synchronously notify watchers of an object.

This blocks until all watchers of the object have received and reacted to the notify, or a timeout is reached.

---

**Note:** BUG: the timeout is not changeable via the C API

BUG: the bufferlist is inaccessible in a rados\_watchcb\_t

---

#### Parameters

- **io** – the pool the object is in
- **o** – the name of the object
- **ver** – obsolete - just pass zero
- **buf** – data to send to watchers
- **buf\_len** – length of buf in bytes

**Returns** 0 on success, negative error code on failure

## 11.2 LibradosPP (C++)

### Todo

write me!

---

## 11.3 Librbd (Python)

The *rbd* python module provides file-like access to RBD images.

### 11.3.1 Example: Creating and writing to an image

To use *rbd*, you must first connect to RADOS and open an IO context:

```
cluster = rados.Rados(conffile='my_ceph.conf')
cluster.connect()
ioctx = cluster.open_ioctx('mypool')
```

Then you instantiate an :class:rbd.RBD object, which you use to create the image:

```
rbd_inst = rbd.RBD()
size = 4 * 1024**3 # 4 GiB
rbd_inst.create(ioctx, 'myimage', size)
```

To perform I/O on the image, you instantiate an `:class:rbd.Image` object:

```
image = rbd.Image(ioctx, 'myimage')
data = 'foo' * 200
image.write(data, 0)
```

This writes 'foo' to the first 600 bytes of the image. Note that data cannot be `:type:unicode` - *Librbd* does not know how to deal with characters wider than a `:c:type:char`.

In the end, you'll want to close the image, the IO context and the connection to RADOS:

```
image.close()
ioctx.close()
cluster.shutdown()
```

To be safe, each of these calls would need to be in a separate `:finally` block:

```
cluster = rados.Rados(conffile='my_ceph_conf')
try:
    ioctx = cluster.open_ioctx('my_pool')
    try:
        rbd_inst = rbd.RBD()
        size = 4 * 1024**3 # 4 GiB
        rbd_inst.create(ioctx, 'myimage', size)
        image = rbd.Image(ioctx, 'myimage')
        try:
            data = 'foo' * 200
            image.write(data, 0)
        finally:
            image.close()
    finally:
        ioctx.close()
finally:
    cluster.shutdown()
```

This can be cumbersome, so the `Rados`, `Ioctx`, and `Image` classes can be used as context managers that close/shutdown automatically (see [PEP 343](#)). Using them as context managers, the above example becomes:

```
with rados.Rados(conffile='my_ceph.conf') as cluster:
    with cluster.open_ioctx('mypool') as ioctx:
        rbd_inst = rbd.RBD()
        size = 4 * 1024**3 # 4 GiB
        rbd_inst.create(ioctx, 'myimage', size)
        with rbd.Image(ioctx, 'myimage') as image:
            data = 'foo' * 200
            image.write(data, 0)
```

## 11.3.2 API Reference

This module is a thin wrapper around `librbd`.

It currently provides all the synchronous methods of `librbd` that do not use callbacks.

Error codes from `librbd` are turned into exceptions that subclass `Error`. Almost all methods may raise `Error` (the base class of all `rbid` exceptions), `PermissionError` and `IOError`, in addition to those documented for the method.

A number of methods have string arguments, which must not be unicode to interact correctly with `librbd`. If unicode is passed to these methods, a `TypeError` will be raised.

**class** `rd.RBD`

This class wraps librbd CRUD functions.

**clone** (*p\_ioctx*, *p\_name*, *p\_snapname*, *c\_ioctx*, *c\_name*, *features=0*, *order=None*)

Clone a parent rbd snapshot into a COW sparse child.

**Parameters**

- **p\_ioctx** – the parent context that represents the parent snap
- **p\_name** – the parent image name
- **p\_snapname** – the parent image snapshot name
- **c\_ioctx** – the child context that represents the new clone
- **c\_name** – the clone (child) name
- **features** (*int*) – bitmask of features to enable; if set, must include layering
- **order** (*int*) – the image is split into (2\*\*order) byte objects

**Raises** `TypeError`

**Raises** `InvalidArgument`

**Raises** `ImageExists`

**Raises** `FunctionNotSupported`

**Raises** `ArgumentOutOfRangeException`

**create** (*ioctx*, *name*, *size*, *order=None*, *old\_format=True*, *features=0*)

Create an rbd image.

**Parameters**

- **ioctx** (`rados.Ioctx`) – the context in which to create the image
- **name** (*str*) – what the image is called
- **size** (*int*) – how big the image is in bytes
- **order** (*int*) – the image is split into (2\*\*order) byte objects
- **old\_format** (*bool*) – whether to create an old-style image that is accessible by old clients, but can't use more advanced features like layering.
- **features** (*int*) – bitmask of features to enable

**Raises** `ImageExists`

**list** (*ioctx*)

List image names.

**Parameters** **ioctx** (`rados.Ioctx`) – determines which RADOS pool is read

**Returns** `list` – a list of image names

**remove** (*ioctx*, *name*)

Delete an RBD image. This may take a long time, since it does not return until every object that comprises the image has been deleted. Note that all snapshots must be deleted before the image can be removed. If there are snapshots left, `ImageHasSnapshots` is raised. If the image is still open, or the watch from a crashed client has not expired, `ImageBusy` is raised.

**Parameters**

- **ioctx** (`rados.Ioctx`) – determines which RADOS pool the image is in

- **name** (*str*) – the name of the image to remove

**Raises** `ImageNotFound`, `ImageBusy`, `ImageHasSnapshots`

**rename** (*ioctx*, *src*, *dest*)

Rename an RBD image.

**Parameters**

- **ioctx** (`rados.Ioctx`) – determines which RADOS pool the image is in
- **src** (*str*) – the current name of the image
- **dest** (*str*) – the new name of the image

**Raises** `ImageNotFound`, `ImageExists`

**version** ()

Get the version number of the `librbd` C library.

**Returns** a tuple of (major, minor, extra) components of the `librbd` version

**class** `rbdd.Image` (*ioctx*, *name*, *snapshot=None*)

This class represents an RBD image. It is used to perform I/O on the image and interact with snapshots.

**Note:** Any method of this class may raise `ImageNotFound` if the image has been deleted.

**close** ()

Release the resources used by this image object.

After this is called, this object should not be used.

**copy** (*dest\_ioctx*, *dest\_name*)

Copy the image to another location.

**Parameters**

- **dest\_ioctx** (`rados.Ioctx`) – determines which pool to copy into
- **dest\_name** (*str*) – the name of the copy

**Raises** `ImageExists`

**create\_snap** (*name*)

Create a snapshot of the image.

**Parameters** **name** (*str*) – the name of the snapshot

**Raises** `ImageExists`

**list\_snaps** ()

Iterate over the snapshots of an image.

**Returns** `SnapIterator`

**read** (*offset*, *length*)

Read data from the image. Raises `InvalidArgument` if part of the range specified is outside the image.

**Parameters**

- **offset** (*int*) – the offset to start reading at
- **length** (*int*) – how many bytes to read

**Returns** *str* - the data read

**Raises** `InvalidArgument`, `IOError`

**remove\_snap** (*name*)

Delete a snapshot of the image.

**Parameters** *name* (*str*) – the name of the snapshot

**Raises** `IOError`

**resize** (*size*)

Change the size of the image.

**Parameters** *size* (*int*) – the new size of the image

**rollback\_to\_snap** (*name*)

Revert the image to its contents at a snapshot. This is a potentially expensive operation, since it rolls back each object individually.

**Parameters** *name* (*str*) – the snapshot to rollback to

**Raises** `IOError`

**set\_snap** (*name*)

Set the snapshot to read from. Writes will raise `ReadOnlyImage` while a snapshot is set. Pass `None` to unset the snapshot (reads come from the current image), and allow writing again.

**Parameters** *name* (*str or None*) – the snapshot to read from, or `None` to unset the snapshot

**stat** ()

Get information about the image. Currently parent pool and parent name are always `-1` and `''`.

**Returns**

dict - contains the following keys:

- *size* (*int*) - the size of the image in bytes
- *obj\_size* (*int*) - the size of each object that comprises the image
- *num\_objs* (*int*) - the number of objects in the image
- *order* (*int*) -  $\log_2(\text{object\_size})$
- *block\_name\_prefix* (*str*) - the prefix of the RADOS objects used to store the image
- *parent\_pool* (*int*) - deprecated
- *parent\_name* (*str*) - deprecated
- see also `:method:format` and `:method:features`

**write** (*data*, *offset*)

Write data to the image. Raises `InvalidArgument` if part of the write would fall outside the image.

**Parameters**

- *data* (*str*) – the data to be written
- *offset* (*int*) – where to start writing data

**Returns** *int* - the number of bytes written

**Raises** `IncompleteWriteError`, `LogicError`, `InvalidArgument`, `IOError`

**class** `rbd.SnapIterator` (*image*)

Iterator over snapshot info for an image.

Yields a dictionary containing information about a snapshot.

Keys are:

- `id` (int) - numeric identifier of the snapshot
- `size` (int) - size of the image at the time of snapshot (in bytes)
- `name` (str) - name of the snapshot

# CEPH SOURCE CODE

You can build Ceph from source by downloading a release or cloning the `ceph` repository at [github](#). If you intend to build Ceph from source, please see the build pre-requisites first. Making sure you have all the pre-requisites will save you time.

## 12.1 Build Prerequisites

Before you can build Ceph source code or Ceph documentation, you need to install several libraries and tools.

---

**Tip:** Check this section to see if there are specific prerequisites for your Linux/Unix distribution.

---

### 12.1.1 Prerequisites for Building Ceph Source Code

Ceph provides `autoconf` and `automake` scripts to get you started quickly. Ceph build scripts depend on the following:

- `autotools-dev`
- `autoconf`
- `automake`
- `cdb`
- `gcc`
- `g++`
- `git`
- `libboost-dev`
- `libedit-dev`
- `libssl-dev`
- `libtool`
- `libfcgi`
- `libfcgi-dev`
- `libfuse-dev`
- `linux-kernel-headers`

- `libcrypto++-dev`
- `libcrypto++`
- `libexpat1-dev`
- `pkg-config`
- `libcurl4-gnutls-dev`

On Ubuntu, execute `sudo apt-get install` for each dependency that isn't installed on your host.

```
sudo apt-get install autotools-dev autoconf automake cdbsgcc g++ git libboost-dev libedit-dev libssl-dev
```

On Debian/Squeeze, execute `aptitude install` for each dependency that isn't installed on your host.

```
aptitude install autotools-dev autoconf automake cdbsgcc g++ git libboost-dev libedit-dev libssl-dev
```

### Ubuntu Requirements

- `uuid-dev`
- `libkeyutils-dev`
- `libgoogle-perftools-dev`
- `libatomic-ops-dev`
- `libaio-dev`
- `libgdata-common`
- `libgdata13`

Execute `sudo apt-get install` for each dependency that isn't installed on your host.

```
sudo apt-get install uuid-dev libkeyutils-dev libgoogle-perftools-dev libatomic-ops-dev libaio-dev libgdata13
```

### Debian

Alternatively, you may also install:

```
aptitude install fakeroot dpkg-dev
aptitude install debhelper cdbsgcc libexpat1-dev libatomic-ops-dev
```

### openSUSE 11.2 (and later)

- `boost-devel`
- `gcc-c++`
- `libedit-devel`
- `libopenssl-devel`
- `fuse-devel` (optional)

Execute `zypper install` for each dependency that isn't installed on your host.

```
zypper install boost-devel gcc-c++ libedit-devel libopenssl-devel fuse-devel
```



## 12.1.2 Prerequisites for Building Ceph Documentation

Ceph utilizes Python's Sphinx documentation tool. For details on the Sphinx documentation tool, refer to: [Sphinx](#). Follow the directions at [Sphinx 1.1.3](#) to install Sphinx. To run Sphinx, with `admin/build-doc`, at least the following are required:

- `python-dev`
- `python-pip`
- `python-virtualenv`
- `libxml2-dev`
- `libxslt-dev`
- `doxygen`
- `ditaa`
- `graphviz`

Execute `sudo apt-get install` for each dependency that isn't installed on your host.

```
sudo apt-get install python-dev python-pip python-virtualenv libxml2-dev libxslt-dev doxygen ditaa graphviz
```

## 12.2 Downloading a Ceph Release Tarball

As Ceph development progresses, the Ceph team releases new versions of the source code. You may download source code tarballs for Ceph releases here:

[Ceph Release Tarballs](#)

## 12.3 Set Up Git

To check out the Ceph source code, you must have `git` installed on your local host.

### 12.3.1 Install Git

To install `git`, execute:

```
sudo apt-get install git
```

You must also have a `github` account. If you do not have a `github` account, go to [github.com](#) and register. Follow the directions for setting up `git` at [Set Up Git](#).

### 12.3.2 Generate SSH Keys

You must generate SSH keys for `github` to clone the Ceph repository. If you do not have SSH keys for `github`, execute:

```
ssh-keygen
```

Get the key to add to your `github` account (the following example assumes you used the default file path):

```
cat .ssh/id_rsa.pub
```

Copy the public key.

### 12.3.3 Add the Key

Go to your `github` account, click on “Account Settings” (i.e., the ‘tools’ icon); then, click “SSH Keys” on the left side navbar.

Click “Add SSH key” in the “SSH Keys” list, enter a name for the key, paste the key you generated, and press the “Add key” button.

## 12.4 Cloning the Ceph Source Code Repository

To clone the source, you must install Git. See Set Up Git for details.

### 12.4.1 Clone the Source

To clone the Ceph source code repository, execute:

```
git clone --recursive https://github.com/ceph/ceph.git
```

Once `git clone` executes, you should have a full copy of the Ceph repository.

---

**Tip:** Make sure you maintain the latest copies of the submodules included in the repository. Running `git status` will tell you if the submodules are out of date:

```
cd ceph
git status
```

If they are, run:

```
git submodule update
```

---

### 12.4.2 Choose a Branch

Once you clone the source code and submodules, your Ceph repository will be on the `master` branch by default, which is the unstable development branch. You may choose other branches too.

- `master`: The unstable development branch.
- `stable`: The bugfix branch.
- `next`: The release candidate branch.

```
git checkout master
```

## 12.5 Building Ceph

Ceph provides build scripts for source code and for documentation.

## 12.5.1 Building Ceph

Ceph provides `automake` and `configure` scripts to streamline the build process. To build Ceph, navigate to your cloned Ceph repository and execute the following:

```
cd ceph
./autogen.sh
./configure
make
```

You can use `make -j` to execute multiple jobs depending upon your system. For example:

```
make -j4
```

To install Ceph locally, you may also use:

```
sudo make install
```

If you install Ceph locally, `make` will place the executables in `usr/local/bin`. You may add the `ceph.conf` file to the `usr/local/bin` directory to run an evaluation environment of Ceph from a single directory.

## 12.5.2 Building Ceph Documentation

Ceph utilizes Python's Sphinx documentation tool. For details on the Sphinx documentation tool, refer to: [Sphinx](#). To build the Ceph documentation, navigate to the Ceph repository and execute the build script:

```
cd ceph
admin/build-doc
```

Once you build the documentation set, you may navigate to the source directory to view it:

```
cd build-doc/output
```

There should be an `/html` directory and a `/man` directory containing documentation in HTML and manpage formats respectively.

## 12.6 Build Ceph Packages

To build packages, you must clone the Ceph repository. You can create installation packages from the latest code using `dpkg-buildpackage` for Debian/Ubuntu or `rpmbuild` for the RPM Package Manager.

---

**Tip:** When building on a multi-core CPU, use the `-j` and the number of cores \* 2. For example, use `-j4` for a dual-core processor to accelerate the build.

---

### 12.6.1 Advanced Package Tool (APT)

To create `.deb` packages for Debian/Ubuntu, ensure that you have cloned the Ceph repository, installed the build prerequisites and installed `debhelper`:

```
sudo apt-get install debhelper
```

Once you have installed `debhelper`, you can build the packages:

```
sudo dpkg-buildpackage
```

For multi-processor CPUs use the `-j` option to accelerate the build.

## 12.6.2 RPM Package Manager

To create `.rpm` packages, ensure that you have cloned the Ceph repository, installed the build prerequisites and installed `rpm-build` and `rpmdevtools`:

```
yum install rpm-build rpmdevtools
```

Once you have installed the tools, setup an RPM compilation environment:

```
rpmdev-setuptree
```

Fetch the source tarball for the RPM compilation environment:

```
wget -P ~/rpmbuild/SOURCES/ http://ceph.com/download/ceph-<version>.tar.gz
```

Build the RPM packages:

```
rpmbuild -tb ~/rpmbuild/SOURCES/ceph-<version>.tar.gz
```

For multi-processor CPUs use the `-j` option to accelerate the build.

## 12.7 Contributing Source Code

If you are making source contributions to the Ceph project, you must be added to the Ceph project on github.

# INTERNAL DEVELOPER DOCUMENTATION

---

**Note:** If you're looking for how to use Ceph as a library from your own software, please see [API Documentation](#).

---

You can start a development mode Ceph cluster, after compiling the source, with:

```
cd src
install -d -m0755 out dev/osd0
./vstart.sh -n -x -l
# check that it's there
./ceph health
```

---

## Todo

vstart is woefully undocumented and full of sharp sticks to poke yourself with.

---

## Mailing list

The official development email list is `ceph-devel@vger.kernel.org`. Subscribe by sending a message to `majordomo@vger.kernel.org` with the line:

```
subscribe ceph-devel
```

in the body of the message.

## Contents

### 13.1 Configuration Management System

The configuration management system exists to provide every daemon with the proper configuration information. The configuration can be viewed as a set of key-value pairs.

**How can the configuration be set? Well, there are several sources:**

- the ceph configuration file, usually named `ceph.conf`
- **command line arguments::** `-debug-ms=1 -debug-pg=10` etc.

- arguments injected at runtime by using injectargs

### 13.1.1 The Configuration File

Most configuration settings originate in the Ceph configuration file.

**How do we find the configuration file? Well, in order, we check:**

- the default locations
- the environment variable CEPH\_CONF
- the command line argument -c

Each stanza of the configuration file describes the key-value pairs that will be in effect for a particular subset of the daemons. The “global” stanza applies to everything. The “mon”, “osd”, and “mds” stanzas specify settings to take effect for all monitors, all osds, and all mds servers, respectively. A stanza of the form mon.\$name, osd.\$name, or mds.\$name gives settings for the monitor, OSD, or MDS of that name, respectively. Configuration values that appear later in the file win over earlier ones.

A sample configuration file can be found in src/sample.ceph.conf.

### 13.1.2 Metavariables

The configuration system allows any configuration value to be substituted into another value using the \$varname syntax, similar to how bash shell expansion works.

**A few additional special metavariables are also defined:**

- \$host: expands to the current hostname
- \$type: expands to one of “mds”, “osd”, “mon”, or “client”
- \$id: expands to the daemon identifier. For osd.0, this would be 0; for mds.a, it would be a; for client.admin, it would be admin.
- \$num: same as \$id
- \$name: expands to \$type.\$id

### 13.1.3 Readin configuration values

There are two ways for Ceph code to get configuration values. One way is to read it directly from a variable named “g\_conf,” or equivalently, “g\_ceph\_ctx->\_conf.” The other is to register an observer that will be called every time the relevant configuration values changes. This observer will be called soon after the initial configuration is read, and every time after that when one of the relevant values changes. Each observer tracks a set of keys and is invoked only when one of the relevant keys changes.

The interface to implement is found in common/config\_obs.h.

**The observer method should be preferred in new code because**

- It is more flexible, allowing the code to do whatever reinitialization needs to be done to implement the new configuration value.
- It is the only way to create a std::string configuration variable that can be changed by injectargs.
- Even for int-valued configuration options, changing the values in one thread while another thread is reading them can lead to subtle and impossible-to-diagnose bugs.

For these reasons, reading directly from `g_conf` should be considered deprecated and not done in new code. Do not ever alter `g_conf`.

### 13.1.4 Changing configuration values

Configuration values can be changed by calling `g_conf->set_val`. After changing the configuration, you should call `g_conf->apply_changes` to re-run all the affected configuration observers. For convenience, you can call `g_conf->set_val_or_die` to make a configuration change which you think should never fail.

`Injectargs`, `parse_argv`, and `parse_env` are three other functions which modify the configuration. Just like with `set_val`, you should call `apply_changes` after calling these functions to make sure your changes get applied.

## 13.2 CephContext

A `CephContext` represents a single view of the Ceph cluster. It comes complete with a configuration, a set of performance counters (`PerfCounters`), and a heartbeat map. You can find more information about `CephContext` in `src/common/ceph_context.h`.

Generally, you will have only one `CephContext` in your application, called `g_ceph_context`. However, in library code, it is possible that the library user will initialize multiple `CephContexts`. For example, this would happen if he called `rados_create` more than once.

A ceph context is required to issue log messages. Why is this? Well, without the `CephContext`, we would not know which log messages were disabled and which were enabled. The `dout()` macro implicitly references `g_ceph_context`, so it can't be used in library code. It is fine to use `dout` and `derr` in daemons, but in library code, you must use `ldout` and `lderr`, and pass in your own `CephContext` object. The compiler will enforce this restriction.

## 13.3 CephFS delayed deletion

When you delete a file, the data is not immediately removed. Each object in the file needs to be removed independently, and sending `size_of_file / stripe_size * replication_count` messages would slow the client down too much, and use a too much of the clients bandwidth. Additionally, snapshots may mean some objects should not be deleted.

Instead, the file is marked as deleted on the MDS, and deleted lazily.

## 13.4 Documenting Ceph

### 13.4.1 Code Documentation

C and C++ can be documented with [Doxygen](#), using the subset of Doxygen markup supported by [Asphyxiate](#).

The general format for function documentation is:

```
/**
 * Short description
 *
 * Detailed description when necessary
 *
 * preconditons, postconditions, warnings, bugs or other notes
 */
```

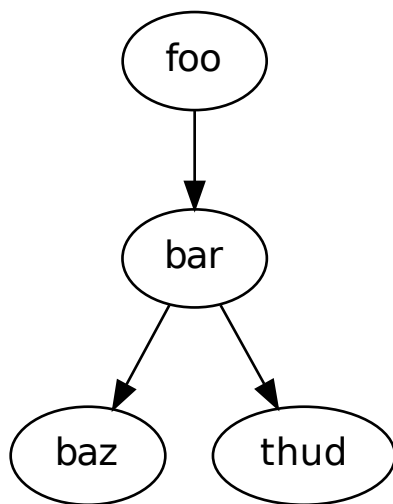
```
* parameter reference
* return value (if non-void)
*/
```

This should be in the header where the function is declared, and functions should be grouped into logical categories. The [librados C API](#) provides a complete example. It is pulled into Sphinx by [librados.rst](#), which is rendered at [Librados \(C\)](#).

## 13.4.2 Drawing diagrams

### Graphviz

You can use [Graphviz](#), as explained in the [Graphviz extension documentation](#).



Most of the time, you'll want to put the actual DOT source in a separate file, like this:

```
.. graphviz:: myfile.dot
```

### Ditaa

You can use [Ditaa](#):

### Blockdiag

If a use arises, we can integrate [Blockdiag](#). It is a Graphviz-style declarative language for drawing things, and includes:

- [block diagrams](#): boxes and arrows (automatic layout, as opposed to [Ditaa](#))
- [sequence diagrams](#): timelines and messages between them
- [activity diagrams](#): subsystems and activities in them



- **network diagrams:** hosts, LANs, IP addresses etc (with [Cisco icons](#) if wanted)

## Inkscape

You can use Inkscape to generate scalable vector graphics. <http://inkscape.org> for restructuredText documents.

If you generate diagrams with Inkscape, you should commit both the Scalable Vector Graphics (SVG) file and export a Portable Network Graphic (PNG) file. Reference the PNG file.

By committing the SVG file, others will be able to update the SVG diagrams using Inkscape.

HTML5 will support SVG inline.

## 13.5 File striping

The text below describes how files from Ceph file system clients are stored across objects stored in RADOS.

### 13.5.1 ceph\_file\_layout

Ceph distributes (stripes) the data for a given file across a number of underlying objects. The way file data is mapped to those objects is defined by the `ceph_file_layout` structure. The data distribution is a modified RAID 0, where data is striped across a set of objects up to a (per-file) fixed size, at which point another set of objects holds the file's data. The second set also holds no more than the fixed amount of data, and then another set is used, and so on.

Defining some terminology will go a long way toward explaining the way file data is laid out across Ceph objects.

- **file** A collection of contiguous data, named from the perspective of the Ceph client (i.e., a file on a Linux system using Ceph storage). The data for a file is divided into fixed-size “stripe units,” which are stored in ceph “objects.”
- **stripe unit** The size (in bytes) of a block of data used in the RAID 0 distribution of a file. All stripe units for a file have equal size. The last stripe unit is typically incomplete—i.e. it represents the data at the end of the file as well as unused “space” beyond it up to the end of the fixed stripe unit size.
- **stripe count** The number of consecutive stripe units that constitute a RAID 0 “stripe” of file data.
- **stripe** A contiguous range of file data, RAID 0 striped across “stripe count” objects in fixed-size “stripe unit” blocks.
- **object** A collection of data maintained by Ceph storage. Objects are used to hold portions of Ceph client files.
- **object set** A set of objects that together represent a contiguous portion of a file.

Three fields in the `ceph_file_layout` structure define this mapping:

```
u32 fl_stripe_unit;
u32 fl_stripe_count;
u32 fl_object_size;
```

(They are actually maintained in their on-disk format, `__le32`.)

The role of the first two fields should be clear from the definitions above.

The third field is the maximum size (in bytes) of an object used to back file data. The object size is a multiple of the stripe unit.

A file's data is blocked into stripe units, and consecutive stripe units are stored on objects in an object set. The number of objects in a set is the same as the stripe count. No object storing file data will exceed the file's designated object size, so after some fixed number of complete stripes, a new object set is used to store subsequent file data.

Note that by default, Ceph uses a simple striping strategy in which `object_size` equals `stripe_unit` and `stripe_count` is 1. This simply puts one `stripe_unit` in each object.

Here's a more complex example:

```
file size = 1 trillion = 1000000000000 bytes
```

```
fl_stripe_unit = 64KB = 65536 bytes
```

```
fl_stripe_count = 5 stripe units per stripe
```

```
fl_object_size = 64GB = 68719476736 bytes
```

This means:

```
file stripe size = 64KB * 5 = 320KB = 327680 bytes
```

```
each object holds 64GB / 64KB = 1048576 stripe units
```

```
file object set size = 64GB * 5 = 320GB = 343597383680 bytes
```

```
(also 1048576 stripe units * 327680 bytes per stripe unit)
```

So the file's 1 trillion bytes can be divided into complete object sets, then complete stripes, then complete stripe units, and finally a single incomplete stripe unit:

- 1 trillion bytes / 320GB per object set = 2 complete object sets  
(with 312805232640 bytes remaining)
- 312805232640 bytes / 320KB per stripe = 954605 complete stripes  
(with 266240 bytes remaining)
- 266240 bytes / 64KB per stripe unit = 4 complete stripe units  
(with 4096 bytes remaining)
- and the final incomplete stripe unit holds those 4096 bytes.

The ASCII art below attempts to capture this:

```

/object 0\ /object 1\ /object 2\ /object 3\ /object 4\
+-----+ +-----+ +-----+ +-----+ +-----+
| stripe | | stripe | | stripe | | stripe | | stripe |
o | unit  | | unit  | | unit  | | unit  | | unit  | stripe 0
b | 0      | | 1      | | 2      | | 3      | | 4      |
j |-----| |-----| |-----| |-----| |-----|
e | stripe | | stripe | | stripe | | stripe | | stripe |
c | unit  | | unit  | | unit  | | unit  | | unit  | stripe 1
t | 5      | | 6      | | 7      | | 8      | | 9      |
|-----| |-----| |-----| |-----| |-----|
s | .      | | .      | | .      | | .      | | .      |
e | .      | | .      | | .      | | .      | | .      |
t | .      | | .      | | .      | | .      | | .      |
|-----| |-----| |-----| |-----| |-----|
0 | stripe | | stripe | | stripe | | stripe | | stripe | stripe
  | unit  | | unit  | | unit  | | unit  | | unit  | 1048575
  | 5242875 | | 5242876 | | 5242877 | | 5242878 | | 5242879 |
\===== / \===== / \===== / \===== / \===== /

```

```

/object 5\ /object 6\ /object 7\ /object 8\ /object 9\
+-----+ +-----+ +-----+ +-----+ +-----+
| stripe | | stripe | | stripe | | stripe | | stripe | stripe
o | unit  | | unit  | | unit  | | unit  | | unit  | 1048576
b | 5242880 | | 5242881 | | 5242882 | | 5242883 | | 5242884 |
j |-----| |-----| |-----| |-----| |-----|
e | stripe | | stripe | | stripe | | stripe | | stripe | stripe
c | unit  | | unit  | | unit  | | unit  | | unit  | 1048577
t | 5242885 | | 5242886 | | 5242887 | | 5242888 | | 5242889 |

```

```

|-----| |-----| |-----| |-----| |-----|
s | . | | . | | . | | . | | . |
e | . | | . | | . | | . | | . |
t | . | | . | | . | | . | | . |
|-----| |-----| |-----| |-----| |-----|
1 | stripe | | stripe | | stripe | | stripe | | stripe | stripe
  | unit | | unit | | unit | | unit | | unit | 2097151
  | 10485755| | 10485756| | 10485757| | 10485758| | 10485759|
  \===== / \===== / \===== / \===== / \===== /

/object 10\ /object 11\ /object 12\ /object 13\ /object 14\
+=====+ +=====+ +=====+ +=====+ +=====+
| stripe | | stripe | | stripe | | stripe | | stripe | stripe
o | unit | | unit | | unit | | unit | | unit | 2097152
b | 10485760| | 10485761| | 10485762| | 10485763| | 10485764|
j |-----| |-----| |-----| |-----| |-----|
e | stripe | | stripe | | stripe | | stripe | | stripe | stripe
c | unit | | unit | | unit | | unit | | unit | 2097153
t | 10485765| | 10485766| | 10485767| | 10485768| | 10485769|
|-----| |-----| |-----| |-----| |-----|
s | . | | . | | . | | . | | . |
e | . | | . | | . | | . | | . |
t | . | | . | | . | | . | | . |
|-----| |-----| |-----| |-----| |-----|
2 | stripe | | stripe | | stripe | | stripe | | stripe | stripe
  | unit | | unit | | unit | | unit | | unit | 3051756
  | 15258780| | 15258781| | 15258782| | 15258783| | 15258784|
  |-----| |-----| |-----| |-----| |-----|
  | stripe | | stripe | | stripe | | stripe | | (partial| (partial
  | unit | | unit | | unit | | unit | | stripe | stripe
  | 15258785| | 15258786| | 15258787| | 15258788| | unit) | 3051757)
  \===== / \===== / \===== / \===== / \===== /

```

## 13.6 Filestore filesystem compatibility

<http://marc.info/?l=ceph-devel&m=131942130322957&w=2>

Although running on ext4, xfs, or whatever other non-btrfs you want mostly works, there are a few important remaining issues:

### 13.6.1 ext4 limits total xattrs for 4KB

This can cause problems in some cases, as Ceph uses xattrs extensively. Most of the time we don't hit this. We do hit the limit with radosgw pretty easily, though, and may also hit it in exceptional cases where the OSD cluster is very unhealthy.

There is a large xattr patch for ext4 from the Lustre folks that has been floating around for (I think) years. Maybe as interest grows in running Ceph on ext4 this can move upstream.

Previously we were being forgiving about large setxattr failures on ext3, but we found that was leading to corruption in certain cases (because we couldn't set our internal metadata), so the next release will assert/crash in that case (fail-stop instead of fail-maybe-eventually-corrupt).

XFS does not have an xattr size limit and thus does have this problem.

## 13.6.2 OSD journal replay of non-idempotent transactions

**Resolved** with full sync but not ideal. See <http://tracker.newdream.net/issues/213>

On non-btrfs backends, the Ceph OSDs use a write-ahead journal. After restart, the OSD does not know exactly which transactions in the journal may have already been committed to disk, and may reapply a transaction again during replay. For most operations (write, delete, truncate) this is fine.

Some operations, though, are non-idempotent. The simplest example is CLONE, which copies (efficiently, on btrfs) data from one object to another. If the source object is modified, the osd restarts, and then the clone is replayed, the target will get incorrect (newer) data. For example,

- clone A -> B
- modify A
- <osd crash, replay from 1>

B will get new instead of old contents.

(This doesn't happen on btrfs because the snapshots allow us to replay from a known consistent point in time.)

Possibilities:

- full sync after any non-idempotent operation
- re-evaluate the lower level interface based on needs from higher levels, construct only safe operations, be very careful; brittle
- use xattrs to add sequence numbers to objects:
  - on non-btrfs, we set a xattr on every modified object with the op\_seq, the unique sequence number for the transaction.
  - for any (potentially) non-idempotent operation, we fsync() before continuing to the next transaction, to ensure that xattr hits disk.
  - on replay, we skip a transaction if the xattr indicates we already performed this transaction.

Because every 'transaction' only modifies on a single object (file), this ought to work. It'll make things like clone slow, but let's face it: they're already slow on non-btrfs file systems because they actually copy the data (instead of duplicating the extent refs in btrfs). And it should make the full ObjectStore interface safe, without upper layers having to worry about the kinds and orders of transactions they perform.

## 13.7 Building Ceph Documentation

Ceph utilizes Python's Sphinx documentation tool. For details on the Sphinx documentation tool, refer to [The Sphinx Documentation Tool](#).

To build the Ceph documentation set, you must:

1. Clone the Ceph repository
2. Install the required tools
3. Build the documents

### 13.7.1 Clone the Ceph Repository

To clone the Ceph repository, you must have `git` installed on your local host. To install `git`, execute:

```
$ sudo apt-get install git
```

You must also have a github account. If you do not have a github account, go to [github](#) and register.

You must set up SSH keys with github to clone the Ceph repository. If you do not have SSH keys for github, execute:

```
$ ssh-keygen -d
```

Get the key to add to your github account:

```
$ cat .ssh/id_dsa.pub
```

Copy the public key. Then, go to your github account, click on **Account Settings** (*i.e.*, the tools icon); then, click **SSH Keys** on the left side navbar.

Click **Add SSH key** in the **SSH Keys** list, enter a name for the key, paste the key you generated, and press the **Add key** button.

To clone the Ceph repository, execute:

```
$ git clone git@github.com:ceph/ceph.git
```

You should have a full copy of the Ceph repository.

### 13.7.2 Install the Required Tools

If you do not have Sphinx and its dependencies installed, a list of dependencies will appear in the output. Install the dependencies on your system, and then execute the build.

To run Sphinx, at least the following are required:

- python-dev
- python-pip
- python-virtualenv
- libxml2-dev
- libxslt-dev
- doxygen
- ditaa
- graphviz

Execute `apt-get install` for each dependency that isn't installed on your host.

```
$ apt-get install python-dev python-pip python-virtualenv libxml2-dev
libxslt-dev doxygen ditaa graphviz
```

### 13.7.3 Build the Documents

Once you have installed all the dependencies, execute the build:

```
$ cd ceph$ admin/build-doc
```

Once you build the documentation set, you may navigate to the source directory to view it:

```
$ cd build-doc/output
```

There should be an `html` directory and a `man` directory containing documentation in HTML and manpage formats respectively.

## 13.8 Kernel client troubleshooting (FS)

If there is an issue with the cephfs kernel client, the most important thing is figuring out whether the problem is with the client or the MDS. Generally, this is easy to work out. If the kernel client broke directly, there will be output in `dmesg`. Collect it and any appropriate kernel state. If the problem is with the MDS, there will be hung requests that the client is waiting on. Look in `/sys/kernel/debug/ceph/*/` and cat the `mdsc` file to get a listing of requests in progress. If one of them remains there, the MDS has probably “forgotten” it. We can get hints about what’s going on by dumping the MDS cache: `ceph mds tell 0 dumpcache /tmp/dump.txt`

And if high logging levels are set on the MDS, that will almost certainly hold the information we need to diagnose and solve the issue.

## 13.9 Library architecture

Ceph is structured into libraries which are built and then combined together to make executables and other libraries.

- `libcommon`: a collection of utilities which are available to nearly every ceph library and executable. In general, `libcommon` should not contain global variables, because it is intended to be linked into libraries such as `libcephfs.so`.
- `libglobal`: a collection of utilities focused on the needs of Ceph daemon programs. In here you will find pidfile management functions, signal handlers, and so forth.

---

### Todo

document other libraries

---

## 13.10 Debug logs

The main debugging tool for Ceph is the `dout` and `derr` logging functions. Collectively, these are referred to as “dout logging.”

`Dout` has several log faculties, which can be set at various log levels using the configuration management system. So it is possible to enable debugging just for the messenger, by setting `debug_ms` to 10, for example.

`Dout` is implemented mainly in `common/DoutStreambuf.cc`

The `dout` macro avoids even generating log messages which are not going to be used, by enclosing them in an “if” statement. What this means is that if you have the debug level set at 0, and you run this code:

```
dout(20) << "myfoo() = " << myfoo() << endl;
```

`myfoo()` will not be called here.

Unfortunately, the performance of debug logging is relatively low. This is because there is a single, process-wide mutex which every debug output statement takes, and every debug output statement leads to a `write()` system call or a call to `syslog()`. There is also a computational overhead to using C++ streams to consider. So you will need to be parsimonious in your logging to get the best performance.

Sometimes, enabling logging can hide race conditions and other bugs by changing the timing of events. Keep this in mind when debugging.

### 13.10.1 Performance counters

Ceph daemons use performance counters to track key statistics like number of inodes pinned. Performance counters are essentially sets of integers and floats which can be set, incremented, and read using the PerfCounters api.

A PerfCounters object is usually associated with a single subsystem. It contains multiple counters. This object is thread-safe because it is protected by an internal mutex. You can create multiple PerfCounters objects.

Currently, three types of performance counters are supported: u64 counters, float counters, and long-run floating-point average counters. These are created by `PerfCountersBuilder::add_u64`, `PerfCountersBuilder::add_fl`, and `PerfCountersBuilder::add_fl_avg`, respectively. u64 and float counters simply provide a single value which can be updated, incremented, and read atomically. floating-point average counters provide two values: the current total, and the number of times the total has been changed. This is intended to provide a long-run average value.

Performance counter information can be read in JSON format from the administrative socket (`admin_sock`). This is implemented as a UNIX domain socket. The Ceph performance counter plugin for `collectd` shows an example of how to access this information. Another example can be found in the unit tests for the administrative sockets.

## 13.11 Monitor bootstrap

Terminology:

- `cluster`: a set of monitors
- `quorum`: an active set of monitors consisting of a majority of the cluster

In order to initialize a new monitor, it must always be fed:

1. a logical name
2. secret keys
3. a cluster fsid (uuid)

In addition, a monitor needs to know two things:

1. what address to bind to
2. who its peers are (if any)

There are a range of ways to do both.

### 13.11.1 Logical id

The logical id should be unique across the cluster. It will be appended to `mon.` to logically describe the monitor in the Ceph cluster. For example, if the logical id is `foo`, the monitor's name will be `mon.foo`.

For most users, there is no more than one monitor per host, which makes the short hostname logical choice.

### 13.11.2 Secret keys

The `mon.` secret key is stored a `keyring` file in the `mon` data directory. It can be generated with a command like:

```
ceph-authtool --create /path/to/keyring --gen-key -n mon.
```

When creating a new monitor cluster, the keyring should also contain a `client.admin` key that can be used to administer the system:

```
ceph-authtool /path/to/keyring --gen-key -n client.admin
```

The resulting keyring is fed to `ceph-mon --mkfs` with the `--keyring <keyring>` command-line argument.

### 13.11.3 Cluster fsid

The cluster fsid is a normal uuid, like that generated by the `uuidgen` command. It can be provided to the monitor in two ways:

1. via the `--fsid <uuid>` command-line argument (or config file option)
2. via a monmap provided to the new monitor via the `--monmap <path>` command-line argument.

### 13.11.4 Monitor address

The monitor address can be provided in several ways.

1. via the `--public-addr <ip[:port]>` command-line option (or config file option)
2. via the `--public-network <cidr>` command-line option (or config file option)
3. via the monmap provided via `--monmap <path>`, if it includes a monitor with our name
4. via the bootstrap monmap (provided via `--monmap <path>` or generated from `--mon-host <list>`) if it includes a monitor with no name (`noame-<something>`) and an address configured on the local host.

### 13.11.5 Peers

The monitor peers are provided in several ways:

1. via the initial monmap, provided via `--monmap <filename>`
2. via the bootstrap monmap generated from `--mon-host <list>`
3. via the bootstrap monmap generated from `[mon.*]` sections with `mon addr` in the config file
4. dynamically via the admin socket

However, these methods are not completely interchangeable because of the complexity of creating a new monitor cluster without danger of races.

### 13.11.6 Cluster creation

There are three basic approaches to creating a cluster:

1. Create a new cluster by specifying the monitor names and addresses ahead of time.
2. Create a new cluster by specifying the monitor names ahead of time, and dynamically setting the addresses as `ceph-mon` daemons configure themselves.
3. Create a new cluster by specifying the monitor addresses ahead of time.



## Names and addresses

Generate a monmap using `monmaptool` with the names and addresses of the initial monitors. The generated monmap will also include a cluster fsid. Feed that monmap to each monitor daemon:

```
ceph-mon --mkfs -i <name> --monmap <initial_monmap> --keyring <initial_keyring>
```

When the daemons start, they will know exactly who they and their peers are.

## Addresses only

The initial monitor addresses can be specified with the `mon host` configuration value, either via a config file or the command-line argument. This method has the advantage that a single global config file for the cluster can have a line like:

```
mon host = a.foo.com, b.foo.com, c.foo.com
```

and will also serve to inform any ceph clients or daemons who the monitors are.

The `ceph-mon` daemons will need to be fed the initial keyring and cluster fsid to initialize themselves:

```
ceph-mon --mkfs -i <name> --fsid <uuid> --keyring <initial_keyring>
```

When the daemons first start up, they will share their names with each other and form a new cluster.

## Names only

In dynamic “cloud” environments, the cluster creator may not (yet) know what the addresses of the monitors are going to be. Instead, they may want machines to configure and start themselves in parallel and, as they come up, form a new cluster on their own. The problem is that the monitor cluster relies on strict majorities to keep itself consistent, and in order to “create” a new cluster, it needs to know what the *initial* set of monitors will be.

This can be done with the `mon initial members` config option, which should list the ids of the initial monitors that are allowed to create the cluster:

```
mon initial members = foo, bar, baz
```

The monitors can then be initialized by providing the other pieces of information (they keyring, cluster fsid, and a way of determining their own address). For example:

```
ceph-mon --mkfs -i <name> --mon-initial-hosts 'foo,bar,baz' --keyring <initial_keyring> --public-add
```

When these daemons are started, they will know their own address, but not their peers. They can learn those addresses via the admin socket:

```
ceph --admin-daemon /var/run/ceph/mon.<id>.asok add_bootstrap_peer_hint <peer ip>
```

Once they learn enough of their peers from the initial member set, they will be able to create the cluster.

## 13.11.7 Cluster expansion

Cluster expansion is slightly less demanding than creation, because the creation of the initial quorum is not an issue and there is no worry about creating separately independent clusters.

New nodes can be forced to join an existing cluster in two ways:

1. by providing no initial monitor peers addresses, and feeding them dynamically.

2. by specifying the `mon initial members config` option to prevent the new nodes from forming a new, independent cluster, and feeding some existing monitors via any available method.

## Initially peerless expansion

Create a new monitor and give it no peer addresses other than it's own. For example:

```
ceph-mon --mkfs -i <myid> --fsid <fsid> --keyring <mon secret key> --public-addr <ip>
```

Once the daemon starts, you can give it one or more peer addresses to join with:

```
ceph --admin-daemon /var/run/ceph/mon.<id>.asok add_bootstrap_peer_hint <peer ip>
```

This monitor will never participate in cluster creation; it can only join an existing cluster.

## Expanding with initial members

You can feed the new monitor some peer addresses initially and avoid badness by also setting `mon initial members`. For example:

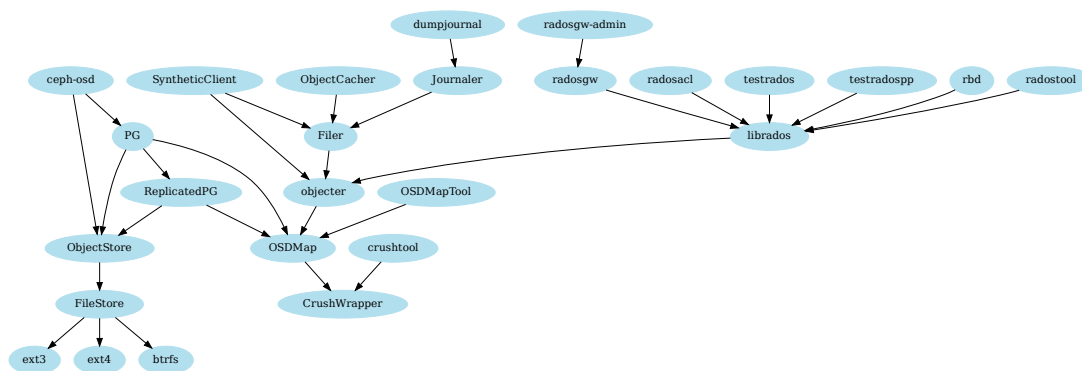
```
ceph-mon --mkfs -i <myid> --fsid <fsid> --keyring <mon secret key> --public-addr <ip> --mon-host foo,
```

When the daemon is started, `mon initial` members must be set via the command line or config file:

```
ceph-mon -i <myid> --mon-initial-members foo,bar,baz
```

to prevent any risk of split-brain.

## 13.12 Object Store Architecture Overview



## Todo

write more here

## 13.13 OSD class path issues

2011-12-05 17:41:00.994075 7ffe8b5c3760 librbd: failed to assign a block name for image  
create error: error 5: Input/output error

This usually happens because your osds can't find `cls_rbd.so`. They search for it in `osd_class_dir`, which may not be set correctly by default (<http://tracker.newdream.net/issues/1722>).

Most likely it's looking in `/usr/lib/rados-classes` instead of `/usr/lib64/rados-classes` - change `osd_class_dir` in your `ceph.conf` and restart the osds to fix it.

## 13.14 Peering

### 13.14.1 Concepts

**Peering** the process of bringing all of the OSDs that store a Placement Group (PG) into agreement about the state of all of the objects (and their metadata) in that PG. Note that agreeing on the state does not mean that they all have the latest contents.

**Acting set** the ordered list of OSDs who are (or were as of some epoch) responsible for a particular PG.

**Up set** the ordered list of OSDs responsible for a particular PG for a particular epoch according to CRUSH. Normally this is the same as the *acting set*, except when the *acting set* has been explicitly overridden via `pg_temp` in the OSDMap.

**current interval or past interval** a sequence of osd map epochs during which the *acting set* and *up set* for particular PG do not change

**primary** the (by convention first) member of the *acting set*, who is responsible for coordination peering, and is the only OSD that will accept client initiated writes to objects in a placement group.

**replica** a non-primary OSD in the *acting set* for a placement group (and who has been recognized as such and *activated* by the primary).

**stray** an OSD who is not a member of the current *acting set*, but has not yet been told that it can delete its copies of a particular placement group.

**recovery** ensuring that copies of all of the objects in a PG are on all of the OSDs in the *acting set*. Once *peering* has been performed, the primary can start accepting write operations, and *recovery* can proceed in the background.

**PG info basic metadata about the PG's creation epoch, the version** for the most recent write to the PG, *last epoch started*, *last epoch clean*, and the beginning of the *current interval*. Any inter-OSD communication about PGs includes the *PG info*, such that any OSD that knows a PG exists (or once existed) also has a lower bound on *last epoch clean* or *last epoch started*.

**PG log** a list of recent updates made to objects in a PG. Note that these logs can be truncated after all OSDs in the *acting set* have acknowledged up to a certain point.

**missing set** Each OSD notes update log entries and if they imply updates to the contents of an object, adds that object to a list of needed updates. This list is called the *missing set* for that <OSD,PG>.

**Authoritative History** a complete, and fully ordered set of operations that, if performed, would bring an OSD's copy of a Placement Group up to date.

**epoch** a (monotonically increasing) OSD map version number

**last epoch start** the last epoch at which all nodes in the *acting set* for a particular placement group agreed on an *authoritative history*. At this point, *peering* is deemed to have been successful.

***up\_thru*** before a primary can successfully complete the *peering* process, it must inform a monitor that is alive through the current osd map epoch by having the monitor set its *up\_thru* in the osd map. This helps peering ignore previous *acting sets* for which peering never completed after certain sequences of failures, such as the second interval below:

- *acting set* = [A,B]
- *acting set* = [A]
- *acting set* = [] very shortly after (e.g., simultaneous failure, but staggered detection)
- *acting set* = [B] (B restarts, A does not)

***last epoch clean*** the last epoch at which all nodes in the *acting set* for a particular placement group were completely up to date (both PG logs and object contents). At this point, *recovery* is deemed to have been completed.

### 13.14.2 Description of the Peering Process

The *Golden Rule* is that no write operation to any PG is acknowledged to a client until it has been persisted by all members of the *acting set* for that PG. This means that if we can communicate with at least one member of each *acting set* since the last successful *peering*, someone will have a record of every (acknowledged) operation since the last successful *peering*. This means that it should be possible for the current primary to construct and disseminate a new *authoritative history*.

It is also important to appreciate the role of the OSD map (list of all known OSDs and their states, as well as some information about the placement groups) in the *peering* process:

When OSDs go up or down (or get added or removed) this has the potential to affect the *active sets* of many placement groups.

Before a primary successfully completes the *peering* process, the osd map must reflect that the OSD was alive and well as of the first epoch in the *current interval*.

Changes can only be made after successful *peering*.

Thus, a new primary can use the latest OSD map along with a recent history of past maps to generate a set of *past intervals* to determine which OSDs must be consulted before we can successfully *peer*. The set of past intervals is bounded by *last epoch started*, the most recent *past interval* for which we know *peering* completed. The process by which an OSD discovers a PG exists in the first place is by exchanging *PG info* messages, so the OSD always has some lower bound on *last epoch started*.

The high level process is for the current PG primary to:

1. get a recent OSD map (to identify the members of the all interesting *acting sets*, and confirm that we are still the primary).
2. generate a list of *past intervals* since *last epoch started*. Consider the subset of those for which *up\_thru* was greater than the first interval epoch by the last interval epoch's osd map; that is, the subset for which *peering* could have completed before the *acting set* changed to another set of OSDs.

Successful *peering* will require that we be able to contact at least one OSD from each of *past interval's acting set*.

3. ask every node in that list for its *PG info*, which includes the most recent write made to the PG, and a value for *last epoch started*. If we learn about a *last epoch started* that is newer than our own, we can prune older *past intervals* and reduce the peer OSDs we need to contact.

5. if anyone else has (in his PG log) operations that I do not have, instruct them to send me the missing log entries so that the primary's *PG log* is up to date (includes the newest write)..
5. for each member of the current *acting set*:
  - (a) ask him for copies of all PG log entries since *last epoch start* so that I can verify that they agree with mine (or know what objects I will be telling him to delete).
 

If the cluster failed before an operation was persisted by all members of the *acting set*, and the subsequent *peering* did not remember that operation, and a node that did remember that operation later rejoined, his logs would record a different (divergent) history than the *authoritative history* that was reconstructed in the *peering* after the failure.

Since the *divergent* events were not recorded in other logs from that *acting set*, they were not acknowledged to the client, and there is no harm in discarding them (so that all OSDs agree on the *authoritative history*). But, we will have to instruct any OSD that stores data from a divergent update to delete the affected (and now deemed to be apocryphal) objects.
  - (b) ask him for his *missing set* (object updates recorded in his PG log, but for which he does not have the new data). This is the list of objects that must be fully replicated before we can accept writes.
6. at this point, the primary's PG log contains an *authoritative history* of the placement group, and the OSD now has sufficient information to bring any other OSD in the *acting set* up to date.
7. if the primary's *up\_thru* value in the current OSD map is not greater than or equal to the first epoch in the *current interval*, send a request to the monitor to update it, and wait until receive an updated OSD map that reflects the change.
8. for each member of the current *acting set*:
  - (a) send them log updates to bring their PG logs into agreement with my own (*authoritative history*) ... which may involve deciding to delete divergent objects.
  - (b) await acknowledgement that they have persisted the PG log entries.
9. at this point all OSDs in the *acting set* agree on all of the meta-data, and would (in any future *peering*) return identical accounts of all updates.
  - (a) start accepting client write operations (because we have unanimous agreement on the state of the objects into which those updates are being accepted). Note, however, that if a client tries to write to an object it will be promoted to the front of the recovery queue, and the write will be applied after it is fully replicated to the current *acting set*.
  - (b) update the *last epoch started* value in our local *PG info*, and instruct other *active set* OSDs to do the same.
  - (c) start pulling object data updates that other OSDs have, but I do not. We may need to query OSDs from additional *past intervals* prior to *last epoch started* (the last time *peering* completed) and following *last epoch clean* (the last epoch that recovery completed) in order to find copies of all objects.
  - (d) start pushing object data updates to other OSDs that do not yet have them.
 

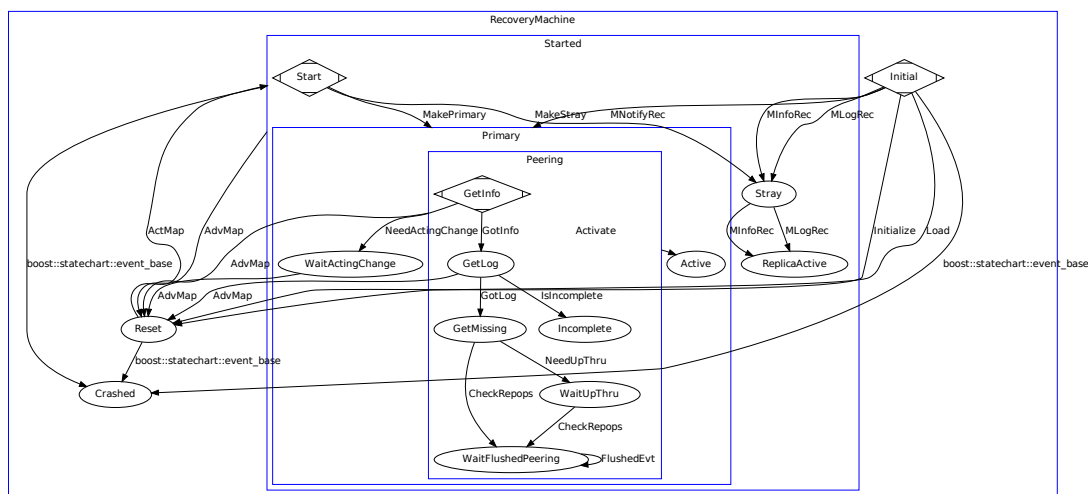
We push these updates from the primary (rather than having the replicas pull them) because this allows the primary to ensure that a replica has the current contents before sending it an update write. It also makes it possible for a single read (from the

primary) to be used to write the data to multiple replicas. If each replica did its own pulls, the data might have to be read multiple times.

10. once all replicas store the all copies of all objects (that existed prior to the start of this epoch) we can update *last epoch clean* in the *PG info*, and we can dismiss all of the *stray* replicas, allowing them to delete their copies of objects for which they are no longer in the *acting set*.

We could not dismiss the *strays* prior to this because it was possible that one of those *strays* might hold the sole surviving copy of an old object (all of whose copies disappeared before they could be replicated on members of the current *acting set*).

### 13.14.3 State Model



## 13.15 Perf counters

The perf counters provide generic internal infrastructure for gauges and counters. The counted values can be both integer and float. There is also an “average” type (normally float) that combines a sum and num counter which can be divided to provide an average.

The intention is that this data will be collected and aggregated by a tool like `collectd` or `statsd` and fed into a tool like `graphite` for graphing and analysis.

### 13.15.1 Access

The perf counter data is accessed via the admin socket. For example:

```
ceph --admin-daemon /var/run/ceph/ceph-osd.0.asok perf schema
ceph --admin-daemon /var/run/ceph/ceph-osd.0.asok perf dump
```

### 13.15.2 Collections

The values are grouped into named collections, normally representing a subsystem or an instance of a subsystem. For example, the internal `throttle` mechanism reports statistics on how it is throttling, and each instance is named something like:

```
throttle-msgr_dispatch_throttler-hbserver
throttle-msgr_dispatch_throttler-client
throttle-filestore_bytes
...
```

### 13.15.3 Schema

The `perf schema` command dumps a json description of which values are available, and what their type is. Each named value as a type bitfield, with the following bits defined.

| bit | meaning                       |
|-----|-------------------------------|
| 1   | floating point value          |
| 2   | unsigned 64-bit integer value |
| 4   | average (sum + count pair)    |
| 8   | counter (vs gauge)            |

Every value will have either bit 1 or 2 set to indicate the type (float or integer). If bit 8 is set (counter), the reader may want to subtract off the previously read value to get the delta during the previous interval.

If bit 4 is set (average), there will be two values to read, a sum and a count. If it is a counter, the average for the previous interval would be sum delta (since the previous read) divided by the count delta. Alternatively, dividing the values outright would provide the lifetime average value. Normally these are used to measure latencies (number of requests and a sum of request latencies), and the average for the previous interval is what is interesting.

Here is an example of the schema output:

```
{
  "throttle-msgr_dispatch_throttler-hbserver" : {
    "get_or_fail_fail" : {
      "type" : 10
    },
    "get_sum" : {
      "type" : 10
    },
    "max" : {
      "type" : 10
    },
    "put" : {
      "type" : 10
    },
    "val" : {
      "type" : 10
    },
    "take" : {
      "type" : 10
    },
    "get_or_fail_success" : {
      "type" : 10
    },
    "wait" : {
      "type" : 5
    },
  },
}
```

```
    "get" : {
      "type" : 10
    },
    "take_sum" : {
      "type" : 10
    },
    "put_sum" : {
      "type" : 10
    }
  },
  "throttle-msgr_dispatch_throttler-client" : {
    "get_or_fail_fail" : {
      "type" : 10
    },
    "get_sum" : {
      "type" : 10
    },
    "max" : {
      "type" : 10
    },
    "put" : {
      "type" : 10
    },
    "val" : {
      "type" : 10
    },
    "take" : {
      "type" : 10
    },
    "get_or_fail_success" : {
      "type" : 10
    },
    "wait" : {
      "type" : 5
    },
    "get" : {
      "type" : 10
    },
    "take_sum" : {
      "type" : 10
    },
    "put_sum" : {
      "type" : 10
    }
  }
}
```

### 13.15.4 Dump

The actual dump is similar to the schema, except that average values are grouped. For example:

```
{
  "throttle-msgr_dispatch_throttler-hbserver" : {
    "get_or_fail_fail" : 0,
    "get_sum" : 0,
    "max" : 104857600,
    "put" : 0,
```



```

    "val" : 0,
    "take" : 0,
    "get_or_fail_success" : 0,
    "wait" : {
        "avgcount" : 0,
        "sum" : 0
    },
    "get" : 0,
    "take_sum" : 0,
    "put_sum" : 0
},
"throttle-msgr_dispatch_throttler-client" : {
    "get_or_fail_fail" : 0,
    "get_sum" : 82760,
    "max" : 104857600,
    "put" : 2637,
    "val" : 0,
    "take" : 0,
    "get_or_fail_success" : 0,
    "wait" : {
        "avgcount" : 0,
        "sum" : 0
    },
    "get" : 2637,
    "take_sum" : 0,
    "put_sum" : 82760
}
}

```

## 13.16 PG (Placement Group) notes

Miscellaneous copy-pastes from emails, when this gets cleaned up it should move out of /dev.

### 13.16.1 Overview

PG = “placement group”. When placing data in the cluster, objects are mapped into PGs, and those PGs are mapped onto OSDs. We use the indirection so that we can group objects, which reduces the amount of per-object metadata we need to keep track of and processes we need to run (it would be prohibitively expensive to track eg the placement history on a per-object basis). Increasing the number of PGs can reduce the variance in per-OSD load across your cluster, but each PG requires a bit more CPU and memory on the OSDs that are storing it. We try and ballpark it at 100 PGs/OSD, although it can vary widely without ill effects depending on your cluster. You hit a bug in how we calculate the initial PG number from a cluster description.

There are a couple of different categories of PGs; the 6 that exist (in the original emailer’s `ceph -s` output) are “local” PGs which are tied to a specific OSD. However, those aren’t actually used in a standard Ceph configuration.

### 13.16.2 Mapping algorithm (simplified)

```

> How does the Object->PG mapping look like, do you map more than one object on
> one PG, or do you sometimes map an object to more than one PG? How about the
> mapping of PGs to OSDs, does one PG belong to exactly one OSD?
>

```

> Does one PG represent a fixed amount of storage space?

Many objects map to one PG.

Each object maps to exactly one PG.

One PG maps to a single list of OSDs, where the first one in the list is the primary and the rest are replicas.

Many PGs can map to one OSD.

A PG represents nothing but a grouping of objects; you configure the number of PGs you want (see [http://ceph.com/wiki/Changing\\_the\\_number\\_of\\_PGs](http://ceph.com/wiki/Changing_the_number_of_PGs)), number of OSDs \* 100 is a good starting point, and all of your stored objects are pseudo-randomly evenly distributed to the PGs. So a PG explicitly does NOT represent a fixed amount of storage; it represents 1/pg\_num 'th of the storage you happen to have on your OSDs.

Ignoring the finer points of CRUSH and custom placement, it goes something like this in pseudocode:

```
locator = object_name
obj_hash = hash(locator)
pg = obj_hash % num_pg
osds_for_pg = crush(pg)  # returns a list of osds
primary = osds_for_pg[0]
replicas = osds_for_pg[1:]
```

If you want to understand the crush() part in the above, imagine a perfectly spherical datacenter in a vacuum ;) that is, if all osds have weight 1.0, and there is no topology to the data center (all OSDs are on the top level), and you use defaults, etc, it simplifies to consistent hashing; you can think of it as:

```
def crush(pg):
    all_osds = ['osd.0', 'osd.1', 'osd.2', ...]
    result = []
    # size is the number of copies; primary+replicas
    while len(result) < size:
        r = hash(pg)
        chosen = all_osds[ r % len(all_osds) ]
        if chosen in result:
            # osd can be picked only once
            continue
        result.append(chosen)
    return result
```

### 13.16.3 User-visible PG States

---

#### Todo

diagram of states and how they can overlap

---

**creating** the PG is still being created

**active** requests to the PG will be processed

**clean** all objects in the PG are replicated the correct number of times

**down** a replica with necessary data is down, so the pg is offline

**replay** the PG is waiting for clients to replay operations after an OSD crashed

**splitting** the PG is being split into multiple PGs (not functional as of 2012-02)

**scrubbing** the PG is being checked for inconsistencies

**degraded** some objects in the PG are not replicated enough times yet

**inconsistent** replicas of the PG are not consistent (e.g. objects are the wrong size, objects are missing from one replica after recovery finished, etc.)

**peering** the PG is undergoing the [Peering](#) process

**repair** the PG is being checked and any inconsistencies found will be repaired (if possible)

**recovering** objects are being migrated/synchronized with replicas

**backfill** a special case of recovery, in which the entire contents of the PG are scanned and synchronized, instead of inferring what needs to be transferred from the PG logs of recent operations

**incomplete** a pg is missing a necessary period of history from its log. If you see this state, report a bug, and try to start any failed OSDs that may contain the needed information.

**stale** the PG is in an unknown state - the monitors have not received an update for it since the PG mapping changed.

**remapped** the PG is temporarily mapped to a different set of OSDs from what CRUSH specified

## 13.17 RBD Layering

RBD layering refers to the creation of copy-on-write clones of block devices. This allows for fast image creation, for example to clone a golden master image of a virtual machine into a new instance. To simplify the semantics, you can only create a clone of a snapshot - snapshots are always read-only, so the rest of the image is unaffected, and there's no possibility of writing to them accidentally.

From a user's perspective, a clone is just like any other rbd image. You can take snapshots of them, read/write them, resize them, etc. There are no restrictions on clones from a user's viewpoint.

Note: the terms *child* and *parent* below mean an rbd image created by cloning, and the rbd image snapshot a child was cloned from.

### 13.17.1 Command line interface

Before cloning a snapshot, you must mark it as protected, to prevent it from being deleted while child images refer to it:

```
$ rbd snap protect pool/image@snap
```

Then you can perform the clone:

```
$ rbd clone [--parent] pool/parent@snap [--image] pool2/child1
```

You can create a clone with different object sizes from the parent:

```
$ rbd clone --order 25 pool/parent@snap pool2/child2
```

To delete the parent, you must first mark it unprotected, which checks that there are no children left:

```
$ rbd snap unprotect pool/image@snap
Cannot unprotect: Still in use by pool2/image2
$ rbd children pool/image@snap
pool2/child1
pool2/child2
$ rbd flatten pool2/child1
```

```
$ rbd rm pool2/child2
$ rbd snap rm pool/image@snap
Cannot remove a protected snapshot: pool/image@snap
$ rbd unprotect pool/image@snap
```

Then the snapshot can be deleted like normal:

```
$ rbd snap rm pool/image@snap
```

## 13.17.2 Implementation

### Data Flow

In the initial implementation, called ‘trivial layering’, there will be no tracking of which objects exist in a clone. A read that hits a non-existent object will attempt to read from the parent snapshot, and this will continue recursively until an object exists or an image with no parent is found. This is done through the normal read path from the parent, so differing object sizes between parents and children do not matter.

Before a write to an object is performed, the object is checked for existence. If it doesn’t exist, a copy-up operation is performed, which means reading the relevant range of data from the parent snapshot and writing it (plus the original write) to the child image. To prevent races with multiple writes trying to copy-up the same object, this copy-up operation will include an atomic create. If the atomic create fails, the original write is done instead. This copy-up operation is implemented as a class method so that extra metadata can be stored by it in the future. In trivial layering, the copy-up operation copies the entire range needed to the child object (that is, the full size of the child object). A future optimization could make this copy-up more fine-grained.

Another future optimization could be storing a bitmap of which objects actually exist in a child. This would obviate the check for existence before each write, and let reads go directly to the parent if needed.

These optimizations are discussed in:

<http://marc.info/?l=ceph-devel&m=129867273303846>

### Parent/Child relationships

Children store a reference to their parent in their header, as a tuple of (pool id, image id, snapshot id). This is enough information to open the parent and read from it.

In addition to knowing which parent a given image has, we want to be able to tell if a protected snapshot still has children. This is accomplished with a new per-pool object, *rbd\_children*, which maps (parent pool id, parent image id, parent snapshot id) to a list of child image ids. This is stored in the same pool as the child image because the client creating a clone already has read/write access to everything in this pool, but may not have write access to the parent’s pool. This lets a client with read-only access to one pool clone a snapshot from that pool into a pool they have full access to. It increases the cost of unprotecting an image, since this needs to check for children in every pool, but this is a rare operation. It would likely only be done before removing old images, which is already much more expensive because it involves deleting every data object in the image.

### Protection

Internally, *protection\_state* is a field in the header object that can be in three states. “protected”, “unprotected”, and “unprotecting”. The first two are set as the result of “rbd protect/unprotect”. The “unprotecting” state is set while the “rbd unprotect” command checks for any child images. Only snapshots in the “protected” state may be cloned, so the “unprotected” state prevents a race like:

1. A: walk through all pools, look for clones, find none

2. B: create a clone
3. A: unprotect parent
4. A: `rbd snap rm pool/parent@snap`

## Resizing

Resizing an rbd image is like truncating a sparse file. New space is treated as zeroes, and shrinking an rbd image deletes the contents beyond the old bounds. This means that if you have a 10G image full of data, and you resize it down to 5G and then up to 10G again, the last 5G is treated as zeroes (and any objects that held that data were removed when the image was shrunk).

Layering complicates this because the absence of an object no longer implies it should be treated as zeroes - if the object is part of a clone, it may mean that some data needs to be read from the parent.

To preserve the resizing behavior for clones, we need to keep track of which objects could be stored in the parent. We can track this as the amount of overlap the child has with the parent, since resizing only changes the end of an image. When a child is created, its overlap is the size of the parent snapshot. On each subsequent resize, the overlap is  $\min(\text{overlap}, \text{new\_size})$ . That is, shrinking the image may shrink the overlap, but increasing the image's size does not change the overlap.

Objects that do not exist past the overlap are treated as zeroes. Objects that do not exist before that point fall back to reading from the parent.

Since this overlap changes over time, we store it as part of the metadata for a snapshot as well.

## Renaming

Currently the rbd header object (that stores all the metadata about an image) is named after the name of the image. This makes renaming disrupt clients who have the image open (such as children reading from a parent). To avoid this, we can name the header object by the id of the image, which does not change. That is, the name of the header object could be `rbd_header.$id`, where `$id` is a unique id for the image in the pool.

When a client opens an image, all it knows is the name. There is already a per-pool `rbd_directory` object that maps image names to ids, but if we relied on it to get the id, we could not open any images in that pool if that single object was unavailable. To avoid this dependency, we can store the id of an image in an object called `rbd_id.$image_name`, where `$image_name` is the name of the image. The per-pool `rbd_directory` object is still useful for listing all images in a pool, however.

### 13.17.3 Header changes

The header needs a few new fields:

- `int64_t parent_pool_id`
- `string parent_image_id`
- `uint64_t parent_snap_id`
- `uint64_t overlap` (how much of the image may be referring to the parent)

These are stored in a “parent” key, which is only present if the image has a parent.

## cls\_rbd

Some new methods are needed:

```
/****** methods on the rbd header *****/
/**
 * Sets the parent and overlap keys.
 * Fails if any of these keys exist, since the image already
 * had a parent.
 */
set_parent(uint64_t pool_id, string image_id, uint64_t snap_id)

/**
 * returns the parent pool id, image id, snap id, and overlap, or -ENOENT
 * if parent_pool_id does not exist or is -1
 */
get_parent(uint64_t snapid)

/**
 * Removes the parent key
 */
remove_parent() // after all parent data is copied to the child

/****** methods on the rbd_children object *****/

add_child(uint64_t parent_pool_id, string parent_image_id,
          uint64_t parent_snap_id, string image_id);
remove_child(uint64_t parent_pool_id, string parent_image_id,
            uint64_t parent_snap_id, string image_id);

/**
 * List ids of a given parent
 */
get_children(uint64_t parent_pool_id, string parent_image_id,
            uint64_t parent_snap_id, uint64_t max_return,
            string start);

/**
 * list parent
 */
get_parents(uint64_t max_return, uint64_t start_pool_id,
            string start_image_id, string start_snap_id);

/****** methods on the rbd_id.$image_name object *****/

set_id(string id)
get_id()

/****** methods on the rbd_directory object *****/

dir_get_id(string name);
dir_get_name(string id);
dir_list(string start_after, uint64_t max_return);
dir_add_image(string name, string id);
dir_remove_image(string name, string id);
dir_rename_image(string src, string dest, string id);
```

Two existing methods will change if the image supports layering:

`snapshot_add` - stores current overlap and `has_parent` with other snapshot metadata (images that don't have layering enabled aren't affected)

`set_size` - will adjust the parent overlap down as needed.

## librbd

Opening a child image opens its parent (and this will continue recursively as needed). This means that an `ImageCtx` will contain a pointer to the parent image context. Differing object sizes won't matter, since reading from the parent will go through the parent image context.

Discard will need to change for layered images so that it only truncates objects, and does not remove them. If we removed objects, we could not tell if we needed to read them from the parent.

A new clone method will be added, which takes the same arguments as `create` except size (size of the parent image is used).

Instead of expanding the `rbd_info` struct, we will break the metadata retrieval into several api calls. Right now, the only users of `rbd_stat()` other than 'rbd info' only use it to retrieve image size.

## 13.18 OSD developer documentation

### Contents

### 13.18.1 Map and PG Message handling

#### Overview

The OSD handles routing incoming messages to PGs, creating the PG if necessary in some cases.

PG messages generally come in two varieties:

1. Peering Messages
2. Ops/SubOps

There are several ways in which a message might be dropped or delayed. It is important that the message delaying does not result in a violation of certain message ordering requirements on the way to the relevant PG handling logic:

1. Ops referring to the same object must not be reordered.
2. Peering messages must not be reordered.
3. Subops must not be reordered.

#### MOSDMap

MOSDMap messages may come from either monitors or other OSDs. Upon receipt, the OSD must perform several tasks:

1. Persist the new maps to the filestore. Several PG operations rely on having access to maps dating back to the last time the PG was clean.
2. Update and persist the superblock.

3. Update OSD state related to the current map.
4. Expose new maps to PG processes via *OSDService*.
5. Remove PGs due to pool removal.
6. Queue dummy events to trigger PG map catchup.

Each PG asynchronously catches up to the currently published map during `process_peering_events` before processing the event. As a result, different PGs may have different views as to the “current” map.

One consequence of this design is that messages containing submessages from multiple PGs (`MOSDPGInfo`, `MOSDPGQuery`, `MOSDPGNotify`) must tag each submessage with the PG’s epoch as well as tagging the message as a whole with the OSD’s current published epoch.

## MOSDPGOp/MOSDPGSubOp

See `OSD::dispatch_op`, `OSD::handle_op`, `OSD::handle_sub_op`

`MOSDPGOp`s are used by clients to initiate rados operations. `MOSDSubOps` are used between OSDs to coordinate most non peering activities including replicating `MOSDPGOp` operations.

`OSD::require_same_or_newer_map` checks that the current `OSDMap` is at least as new as the map epoch indicated on the message. If not, the message is queued in `OSD::waiting_for_osdmap` via `OSD::wait_for_new_map`. Note, this cannot violate the above conditions since any two messages will be queued in order of receipt and if a message is recieved with epoch `e0`, a later message from the same source must be at epoch at least `e0`. Note that two PGs from the same OSD count for these purposes as different sources for single PG messages. That is, messages from different PGs may be reordered.

`MOSDPGOp`s follow the following process:

1. `OSD::handle_op`: validates permissions and crush mapping. See `OSDService::handle_misdirected_op` See `OSD::op_has_sufficient_caps` See `OSD::require_same_or_newer_map`
2. `OSD::enqueue_op`

`MOSDSubOps` follow the following process:

1. `OSD::handle_sub_op` checks that sender is an OSD
2. `OSD::enqueue_op`

`OSD::enqueue_op` calls `PG::queue_op` which checks `can_discard_request` before queueing the op in the `op_queue` and the PG in the `OpWQ`. Note, a single PG may be in the op queue multiple times for multiple ops.

`dequeue_op` is then eventually called on the PG. At this time, the op is popped off of `op_queue` and passed to `PG::do_request`, which checks that the PG map is new enough (`must_delay_op`) and then processes the request.

In summary, the possible ways that an op may wait or be discarded in are:

1. Wait in `waiting_for_osdmap` due to `OSD::require_same_or_newer_map` from `OSD::handle_*`.
2. Discarded in `OSD::can_discard_op` at `enqueue_op`.
3. Wait in `PG::op_waiters` due to `PG::must_delay_request` in `PG::do_request`.
4. Wait in `PG::waiting_for_active` in `do_request` due to `!flushed`.
5. Wait in `PG::waiting_for_active` due to `!active()` in `do_op/do_sub_op`.
6. Wait in `PG::waiting_for_(degraded/missing)` in `do_op`.
7. Wait in `PG::waiting_for_active` due to `scrub_block_writes` in `do_op`

TODO: The above is not a complete list.



## Peering Messages

See `OSD::handle_pg_(notify|info|log|query)`

Peering messages are tagged with two epochs:

1. `epoch_sent`: map epoch at which the message was sent
2. `query_epoch`: map epoch at which the message triggering the message was sent

These are the same in cases where there was no triggering message. We discard a peering message if the message's `query_epoch` if the PG in question has entered a new epoch (See `PG::old_peering_event`, `PG::queue_peering_event`). Notices, infos, notifies, and logs are all handled as `PG::RecoveryMachine` events and are wrapped by `PG::queue_*` by `PG::CephPeeringEvts`, which include the created state machine event along with `epoch_sent` and `query_epoch` in order to generically check `PG::old_peering_message` upon insertion and removal from the queue.

Note, notices, logs, and infos can trigger the creation of a PG. See `OSD::get_or_create_pg`.

## 13.18.2 OSD

### Concepts

**Messenger** See `src/msg/Messenger.h`

Handles sending and receipt of messages on behalf of the OSD. The OSD uses two messengers:

1. `cluster_messenger` - handles traffic to other OSDs, monitors
2. `client_messenger` - handles client traffic

This division allows the OSD to be configured with different interfaces for client and cluster traffic.

**Dispatcher** See `src/msg/Dispatcher.h`

OSD implements the Dispatcher interface. Of particular note is `ms_dispatch`, which serves as the entry point for messages received via either the client or cluster messenger. Because there are two messengers, `ms_dispatch` may be called from at least two threads. The `osd_lock` is always held during `ms_dispatch`.

**WorkQueue** See `src/common/WorkQueue.h`

The `WorkQueue` class abstracts the process of queueing independent tasks for asynchronous execution. Each OSD process contains workqueues for distinct tasks:

1. `OpWQ`: handles ops (from clients) and subops (from other OSDs). Runs in the `op_tp` threadpool.
2. `PeeringWQ`: handles peering tasks and pg map advancement Runs in the `op_tp` threadpool. See Peering
3. `CommandWQ`: handles commands (pg query, etc) Runs in the `command_tp` threadpool.
4. `RecoveryWQ`: handles recovery tasks. Runs in the `recovery_tp` threadpool.
5. `SnapTrimWQ`: handles snap trimming Runs in the `disk_tp` threadpool. See `SnapTrimmer`
6. `ScrubWQ`: handles primary scrub path Runs in the `disk_tp` threadpool. See `Scrub`
7. `ScrubFinalizeWQ`: handles primary scrub finalize Runs in the `disk_tp` threadpool. See `Scrub`
8. `RepScrubWQ`: handles replica scrub path Runs in the `disk_tp` threadpool See `Scrub`
9. `RemoveWQ`: Asynchronously removes old pg directories Runs in the `disk_tp` threadpool See `PGRemoval`

**ThreadPool** See src/common/WorkQueue.h See also above.

There are 4 OSD threadpools:

1. op\_tp: handles ops and subops
2. recovery\_tp: handles recovery tasks
3. disk\_tp: handles disk intensive tasks
4. command\_tp: handles commands

**OSDMap** See src/osd/OSDMap.h

The crush algorithm takes two inputs: a picture of the cluster with status information about which nodes are up/down and in/out, and the pgid to place. The former is encapsulated by the OSDMap. Maps are numbered by *epoch* (epoch\_t). These maps are passed around within the OSD as std::tr1::shared\_ptr<const OSDMap>.

See MapHandling

**PG** See src/osd/PG.\* src/osd/ReplicatedPG.\*

Objects in rados are hashed into *PGs* and *PGs* are placed via crush onto OSDs. The PG structure is responsible for handling requests pertaining to a particular *PG* as well as for maintaining relevant metadata and controlling recovery.

**OSDService** See src/osd/OSD.cc OSDService

The OSDService acts as a broker between PG threads and OSD state which allows PGs to perform actions using OSD services such as workqueues and messengers. This is still a work in progress. Future cleanups will focus on moving such state entirely from the OSD into the OSDService.

## Overview

See src/ceph\_osd.cc

The OSD process represents one leaf device in the crush hierarchy. There might be one OSD process per physical machine, or more than one if, for example, the user configures one OSD instance per disk.

## 13.18.3 PG

### Concepts

**Peering Interval** See PG::start\_peering\_interval. See PG::up\_acting\_affected. See PG::RecoveryState::Reset

A peering interval is a maximal set of contiguous map epochs in which the up and acting sets did not change. PG::RecoveryMachine represents a transition from one interval to another as passing through RecoveryState::Reset. On PG::RecoveryState::AdvMap PG::up\_acting\_affected can cause the pg to transition to Reset.

### Peering Details and Gotchas

For an overview of peering, see Peering.

- PG::flushed defaults to false and is set to false in PG::start\_peering\_interval. Upon transitioning to PG::RecoveryState::Started we send a transaction through the pg op sequencer which, upon complete, sends a FlushedEvt which sets flushed to true. The primary cannot go active until this happens (See PG::RecoveryState::WaitFlushedPeering). Replicas can go active but cannot serve ops (writes or reads). This is necessary because we cannot read our ondisk state until unstable transactions from the previous interval have cleared.

### 13.18.4 PG Removal

See `OSD::_remove_pg`, `OSD::RemoveWQ`

There are two ways for a pg to be removed from an OSD:

1. `MOSDPGRemove` from the primary
2. `OSD::advance_map` finds that the pool has been removed

In either case, our general strategy for removing the pg is to atomically remove the metadata objects (`pg->log_oid`, `pg->biginfo_oid`) and rename the pg collections (`temp`, `HEAD`, and `snap` collections) into removal collections (see `OSD::get_next_removal_coll`). Those collections are then asynchronously removed. We do not do this inline because scanning the collections to remove the objects is an expensive operation. Atomically moving the directories out of the way allows us to proceed as if the pg is fully removed except that we cannot rewrite any of the objects contained in the removal directories until they have been fully removed. PGs partition the object space, so the only case we need to worry about is the same pg being recreated before we have finished removing the objects from the old one.

`OSDService::deleting_pgs` tracks all pgs in the process of being deleted. Each `DeletingState` object in `deleting_pgs` lives while at least one reference to it remains. Each item in `RemoveWQ` carries a reference to the `DeletingState` for the relevant pg such that `deleting_pgs.lookup(pgid)` will return a null ref only if there are no collections currently being deleted for that pg. `DeletingState` allows you to register a callback to be called when the deletion is finally complete. See `PG::start_flush`. We use this mechanism to prevent the pg from being “flushed” until any pending deletes are complete. Metadata operations are safe since we did remove the old metadata objects and we inherit the `osr` from the previous copy of the pg.

Similarly, `OSD::osr_registry` ensures that the `OpSequencers` for those pgs can be reused for a new pg if created before the old one is fully removed, ensuring that operations on the new pg are sequenced properly with respect to operations on the old one.

`OSD::load_pgs()` rebuilds `deleting_pgs` and `osr_registry` when scanning the collections as it finds old removal collections not yet removed.



# MANUAL PAGES

## 14.1 Section 1, executable programs or shell commands

### 14.1.1 obsync – The object synchronizer tool

#### Synopsis

**obsync** [ *options* ] *source-url destination-url*

#### Description

**obsync** is an object synchronizer tool designed to transfer objects between different object storage systems. Similar to **rsync**, you specify a source and a destination, and it will transfer objects between them until the destination has all the objects in the source. Obsync will never modify the source – only the destination.

By default, obsync does not delete anything. However, by specifying `--delete-after` or `--delete-before`, you can ask it to delete objects from the destination that are not in the source.

#### Target types

Obsync supports S3 via `libboto`. To use the `s3` target, your URL should look like this:  
`s3://host-name/bucket-name`

Obsync supports storing files locally via the `file://` target. To use the file target, your URL should look like this:  
`file://directory-name`

Alternately, give no prefix, like this: `./directory-name`

Obsync supports storing files in a RADOS Gateway backend via the `librados` Python bindings. To use the `rgw`` target, your URL should look like this:  
``rgw:ceph-configuration-path:rgw-bucket-name`

#### Options

**-h, -help**

Display a help message

**-n, -dry-run**

Show what would be done, but do not modify the destination.

- c, -create-dest**  
Create the destination if it does not exist.
- delete-before**  
Before copying any files, delete objects in the destination that are not in the source.
- L, -follow-symlinks**  
Follow symlinks when dealing with `file://` targets.
- no-preserve-acls**  
Don't preserve ACLs when copying objects.
- v, -verbose**  
Be verbose.
- V, -more-verbose**  
Be really, really verbose (developer mode)
- x SRC=DST, -xuser SRC=DST**  
Set up a user translation. You can specify multiple user translations with multiple `--xuser` arguments.
- force**  
Overwrite all destination objects, even if they appear to be the same as the source objects.

## Environment variables

- SRC\_AKEY**  
Access key for the source URL
- SRC\_SKEY**  
Secret access key for the source URL
- DST\_AKEY**  
Access key for the destination URL
- DST\_SKEY**  
Secret access key for the destination URL
- AKEY**  
Access key for both source and dest
- SKEY**  
Secret access key for both source and dest
- DST\_CONSISTENCY**  
Set to 'eventual' if the destination is eventually consistent. If the destination is eventually consistent, we may have to retry certain operations multiple times.

## Examples

```
AKEY=... SKEY=... obsync -c -d -v ./backup-directory s3://myhost1/mybucket1
```

Copy objects from backup-directory to mybucket1 on myhost1:

```
SRC_AKEY=... SRC_SKEY=... DST_AKEY=... DST_SKEY=... obsync -c -d -v s3://myhost1/mybucket1 s3://myhost1/mybucket1
```

Copy objects from mybucket1 to mybucket2

## Availability

**obsync** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## 14.2 Section 8, system administration commands

### 14.2.1 ceph – ceph file system control utility

#### Synopsis

```
ceph [ -m monaddr ] [ -w | command ... ]
```

#### Description

**ceph** is a control utility for communicating with the monitor cluster of a running Ceph distributed file system.

There are three basic modes of operation.

#### Interactive mode

To start in interactive mode, no arguments are necessary. Control-d or ‘quit’ will exit.

#### Watch mode

Watch mode shows cluster state changes as they occur. For example:

```
ceph -w
```

#### Command line mode

Finally, to send a single instruction to the monitor cluster (and wait for a response), the command can be specified on the command line.

#### Options

- i** *infile*  
will specify an input file to be passed along as a payload with the command to the monitor cluster. This is only used for specific monitor commands.
- o** *outfile*  
will write any payload returned by the monitor cluster with its reply to outfile. Only specific monitor commands (e.g. `osd getmap`) return a payload.
- c** *ceph.conf*, **-conf**=*ceph.conf*  
Use *ceph.conf* configuration file instead of the default `/etc/ceph/ceph.conf` to determine monitor addresses during startup.
- m** *monaddress[:port]*  
Connect to specified monitor (instead of looking through *ceph.conf*).

## Examples

To grab a copy of the current OSD map:

```
ceph -m 1.2.3.4:6789 osd getmap -o osdmap
```

To get a dump of placement group (PG) state:

```
ceph pg dump -o pg.txt
```

## Monitor commands

A more complete summary of commands understood by the monitor cluster can be found in the wiki, at

<http://ceph.com/docs/control>

## Availability

**ceph** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph*(8), *mkcephfs*(8)

## 14.2.2 ceph-authtool – ceph keyring manipulation tool

### Synopsis

```
ceph-authtool keyringfile [ -l | -list ] [ -C | -create-keyring ] [ -p | -print ] [ -n | -name entityname ] [ -gen-key ] [ -a |  
-add-key base64_key ] [ -caps capfiles ]
```

### Description

**ceph-authtool** is a utility to create, view, and modify a Ceph keyring file. A keyring file stores one or more Ceph authentication keys and possibly an associated capability specification. Each key is associated with an entity name, of the form {*client, mon, mds, osd*}.name.

**WARNING** Ceph provides authentication and protection against man-in-the-middle attacks once secret keys are in place. However, data over the wire is not encrypted, which may include the messages used to configure said keys. The system is primarily intended to be used in trusted environments.

### Options

**-l, -list**

will list all keys and capabilities present in the keyring

**-p, -print**

will print an encoded key for the specified entityname. This is suitable for the `mount -o secret=` argument

**-C, -create-keyring**

will create a new keyring, overwriting any existing keyringfile



- gen-key**  
will generate a new secret key for the specified entityname
- add-key**  
will add an encoded key to the keyring
- cap** subsystem capability  
will set the capability for given subsystem
- caps** capsfile  
will set all of capabilities associated with a given key, for all subsystems

## Capabilities

The subsystem is the name of a Ceph subsystem: `mon`, `mds`, or `osd`.

The capability is a string describing what the given user is allowed to do. This takes the form of a comma separated list of allow clauses with a permission specifier containing one or more of `rw` for read, write, and execute permission. The `allow *` grants full superuser permissions for the given subsystem.

For example:

```
# can read, write, and execute objects
osd = "allow rwx [pool=foo[,bar]][uid=baz[,bay]]"

# can access mds server
mds = "allow"

# can modify cluster state (i.e., is a server daemon)
mon = "allow rwx"
```

A librados user restricted to a single pool might look like:

```
osd = "allow rw pool foo"
```

A client mounting the file system with minimal permissions would need caps like:

```
mds = "allow"

osd = "allow rw pool=data"

mon = "allow r"
```

## Caps file format

The caps file format consists of zero or more key/value pairs, one per line. The key and value are separated by an `=`, and the value must be quoted (with `'` or `"`) if it contains any whitespace. The key is the name of the Ceph subsystem (`osd`, `mds`, `mon`), and the value is the capability string (see above).

## Example

To create a new keyring containing a key for `client.foo`:

```
ceph-authtool -C -n client.foo --gen-key keyring
```

To associate some capabilities with the key (namely, the ability to mount a Ceph filesystem):

```
ceph-authtool -n client.foo --cap mds 'allow' --cap osd 'allow rw pool=data' --cap mon 'allow r' keyring
```

To display the contents of the keyring:

```
ceph-authtool -l keyring
```

When mount a Ceph file system, you can grab the appropriately encoded secret key with:

```
mount -t ceph serverhost:/ mountpoint -o name=foo,secret=`ceph-authtool -p -n client.foo keyring`
```

## Availability

**ceph-authtool** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph*(8)

## 14.2.3 ceph-clsinfo – show class object information

### Synopsis

**ceph-clsinfo** [ *options* ] ... *filename*

### Description

**ceph-clsinfo** can show name, version, and architecture information about a specific class object.

### Options

- n, -name**  
Shows the class name
- v, -version**  
Shows the class version
- a, -arch**  
Shows the class architecture

## Availability

**ceph-clsinfo** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph*(8)

## 14.2.4 ceph-conf – ceph conf file tool

### Synopsis

```
ceph-conf -c conffile --list-all-sections
ceph-conf -c conffile -L
ceph-conf -c conffile -l prefix
ceph-conf key -s section1 ...
ceph-conf [-s section ] --lookup key
ceph-conf [-s section ] key
```

### Description

**ceph-conf** is a utility for getting information about a ceph configuration file. As with most Ceph programs, you can specify which Ceph configuration file to use with the `-c` flag.

### Actions

**ceph-conf** will perform one of the following actions:

`--list-all-sections` or `-L` prints out a list of all the section names in the configuration file.

`--list-sections` or `-l` prints out a list of all the sections that begin with a given prefix. For example, `--list-sections mon` would list all sections beginning with `mon`.

`--lookup` will search the configuration for a given value. By default, the sections that are searched are determined by the Ceph name that we are using. The Ceph name defaults to `client.admin`. It can be specified with `--name`.

For example, if we specify `--name osd.0`, the following sections will be searched: `[osd.0]`, `[osd]`, `[global]`

You can specify additional sections to search with `--section` or `-s`. These additional sections will be searched before the sections that would normally be searched. As always, the first matching entry we find will be returned.

Note: `--lookup` is the default action. If no other actions are given on the command line, we will default to doing a lookup.

### Examples

To find out what value `osd 0` will use for the “osd data” option:

```
ceph-conf -c foo.conf --name osd.0 --lookup "osd data"
```

To find out what value will `mds a` use for the “log file” option:

```
ceph-conf -c foo.conf --name mds.a "log file"
```

To list all sections that begin with `osd`:

```
ceph-conf -c foo.conf -l osd
```

To list all sections:

```
ceph-conf -c foo.conf -L
```

## Availability

**ceph-conf** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph(8)*, *mkcephfs(8)*

## 14.2.5 ceph-debugpack – ceph debug packer utility

### Synopsis

**ceph-debugpack** [ *options* ] *filename.tar.gz*

### Description

**ceph-debugpack** will build a tarball containing various items that are useful for debugging crashes. The resulting tarball can be shared with Ceph developers when debugging a problem.

The tarball will include the binaries for ceph-mds, ceph-osd, and ceph-mon, any log files, the ceph.conf configuration file, any core files we can find, and (if the system is running) dumps of the current osd, mds, and pg maps from the monitor.

### Options

**-c** *ceph.conf*, **-conf**=*ceph.conf*

Use *ceph.conf* configuration file instead of the default */etc/ceph/ceph.conf* to determine monitor addresses during startup.

## Availability

**ceph-debugpack** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph(8)*

## 14.2.6 ceph-dencoder – ceph encoder/decoder utility

### Synopsis

**ceph-dencoder** [commands...]

## Description

**ceph-dencoder** is a utility to encode, decode, and dump ceph data structures. It is used for debugging and for testing inter-version compatibility.

**ceph-dencoder** takes a simple list of commands and performs them in order.

## Commands

### version

Print the version string for the **ceph-dencoder** binary.

### import <file>

Read a binary blob of encoded data from the given file. It will be placed in an in-memory buffer.

### export <file>

Write the contents of the current in-memory buffer to the given file.

### list\_types

List the data types known to this build of **ceph-dencoder**.

### type <name>

Select the given type for future `encode` or `decode` operations.

### decode

Decode the contents of the in-memory buffer into an instance of the previously selected type. If there is an error, report it.

### encode

Encode the contents of the in-memory instance of the previously selected type to the in-memory buffer.

### dump\_json

Print a JSON-formatted description of the in-memory object.

### count\_tests

Print the number of built-in test instances of the previously selected type that **ceph-dencoder** is able to generate.

### select\_test <n>

Select the given build-in test instance as a the in-memory instance of the type.

### get\_features

Print the decimal value of the feature set supported by this version of **ceph-dencoder**. Each bit represents a feature. These correspond to `CEPH_FEATURE_*` defines in `src/include/ceph_features.h`.

### set\_features <f>

Set the feature bits provided to `encode` to *f*. This allows you to encode objects such that they can be understood by old versions of the software (for those types that support it).

## Example

Say you want to examine an attribute on an object stored by `ceph-osd`. You can do:

```
$ cd /mnt/osd.12/current/2.b_head
$ attr -l foo_bar_head_EFE6384B
Attribute "ceph.snapset" has a 31 byte value for foo_bar_head_EFE6384B
Attribute "ceph._" has a 195 byte value for foo_bar_head_EFE6384B
$ attr foo_bar_head_EFE6384B -g ceph._ -q > /tmp/a
$ ceph-dencoder type object_info_t import /tmp/a decode dump_json
{ "oid": { "oid": "foo",
```

```
"key": "bar",
"snapid": -2,
"hash": 4024842315,
"max": 0},
"locator": { "pool": 2,
  "preferred": -1,
  "key": "bar"},
"category": "",
"version": "9'1",
"prior_version": "0'0",
"last_reqid": "client.4116.0:1",
"size": 1681,
"mtime": "2012-02-21 08:58:23.666639",
"lost": 0,
"wrlock_by": "unknown.0.0:0",
"snaps": [],
"truncate_seq": 0,
"truncate_size": 0,
"watchers": {}}
```

## Availability

**ceph-dencoder** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph*(8)

## 14.2.7 ceph-fuse – FUSE-based client for ceph

### Synopsis

**ceph-fuse** [ -m *monaddr:port* ] *mountpoint* [ *fuse options* ]

### Description

**ceph-fuse** is a FUSE (File system in USerspace) client for Ceph distributed file system. It will mount a ceph file system (specified via the -m option for described by ceph.conf (see below) at the specific mount point.

The file system can be unmounted with:

```
fusermount -u mountpoint
```

or by sending SIGINT to the `ceph-fuse` process.

### Options

Any options not recognized by `ceph-fuse` will be passed on to `libfuse`.

**-d**

Detach from console and daemonize after startup.

- c** *ceph.conf*, **-conf**=*ceph.conf*  
Use *ceph.conf* configuration file instead of the default */etc/ceph/ceph.conf* to determine monitor addresses during startup.
- m** *monaddress[:port]*  
Connect to specified monitor (instead of looking through *ceph.conf*).
- r** *root\_directory*  
Use *root\_directory* as the mounted root, rather than the full Ceph tree.

## Availability

**ceph-fuse** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

fusermount(8), *ceph*(8)

## 14.2.8 ceph-mds – ceph metadata server daemon

### Synopsis

```
ceph-mds -i name [[ -hot-standby [rank] ][-journal_check rank]]
```

### Description

**ceph-mds** is the metadata server daemon for the Ceph distributed file system. One or more instances of **ceph-mds** collectively manage the file system namespace, coordinating access to the shared OSD cluster.

Each **ceph-mds** daemon instance should have a unique name. The name is used to identify daemon instances in the *ceph.conf*.

Once the daemon has started, the monitor cluster will normally assign it a logical rank, or put it in a standby pool to take over for another daemon that crashes. Some of the specified options can cause other behaviors.

If you specify *hot-standby* or *journal-check*, you must either specify the rank on the command line, or specify one of the *mds\_standby\_for\_[rank|name]* parameters in the config. The command line specification overrides the config, and specifying the rank overrides specifying the name.

### Options

- f, -foreground**  
Foreground: do not daemonize after startup (run in foreground). Do not generate a pid file. Useful when run via *ceph-run*(8).
- d**  
Debug mode: like *-f*, but also send all log output to *stderr*.
- c** *ceph.conf*, **-conf**=*ceph.conf*  
Use *ceph.conf* configuration file instead of the default */etc/ceph/ceph.conf* to determine monitor addresses during startup.

**-m** *monaddress[:port]*  
Connect to specified monitor (instead of looking through *ceph.conf*).

## Availability

**ceph-mon** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph*(8), *ceph-mon*(8), *ceph-osd*(8)

## 14.2.9 ceph-mon – ceph monitor daemon

### Synopsis

**ceph-mon** -i *monid* [ -mon-data *mondatapath* ]

### Description

**ceph-mon** is the cluster monitor daemon for the Ceph distributed file system. One or more instances of **ceph-mon** form a Paxos part-time parliament cluster that provides extremely reliable and durable storage of cluster membership, configuration, and state.

The *mondatapath* refers to a directory on a local file system storing monitor data. It is normally specified via the *mon data* option in the configuration file.

### Options

- f, -foreground**  
Foreground: do not daemonize after startup (run in foreground). Do not generate a pid file. Useful when run via *ceph-run*(8).
- d**  
Debug mode: like *-f*, but also send all log output to stderr.
- c** *ceph.conf*, **-conf**=*ceph.conf*  
Use *ceph.conf* configuration file instead of the default */etc/ceph/ceph.conf* to determine monitor addresses during startup.
- mkfs**  
Initialize the *mon data* directory with seed information to form and initial ceph file system or to join an existing monitor cluster. Three pieces of information must be provided:
- The cluster fsid. This can come from a monmap (*--monmap <path>*) or explicitly via *--fsid <uuid>*.
  - A list of monitors and their addresses. This list of monitors can come from a monmap (*--monmap <path>*), the *mon host* configuration value (in *ceph.conf* or via *-m host1,host2,...*), or *mon addr* lines in *ceph.conf*. If this monitor is to be part of the initial monitor quorum for a new Ceph cluster, then it must be included in the initial list, matching either the name or address of a monitor in the list. When matching by address, either the *public addr* or *public subnet* options may be used.



- The monitor secret key `mon...` This must be included in the keyring provided via `--keyring <path>`.

**-keyring**

Specify a keyring for use with `--mkfs`.

**Availability**

**ceph-mon** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

**See also**

*ceph(8)*, *ceph-mds(8)*, *ceph-osd(8)*

**14.2.10 ceph-osd – ceph object storage daemon****Synopsis**

**ceph-osd** -i *osdnum* [ `-osd-data` *datapath* ] [ `-osd-journal` *journal* ] [ `-mkfs` ] [ `-mkjournal` ] [ `-mkkey` ]

**Description**

**ceph-osd** is the object storage daemon for the Ceph distributed file system. It is responsible for storing objects on a local file system and providing access to them over the network.

The *datapath* argument should be a directory on a btrfs file system where the object data resides. The journal is optional, and is only useful performance-wise when it resides on a different disk than *datapath* with low latency (ideally, an NVRAM device).

**Options****-f, -foreground**

Foreground: do not daemonize after startup (run in foreground). Do not generate a pid file. Useful when run via *ceph-run(8)*.

**-d**

Debug mode: like `-f`, but also send all log output to stderr.

**-osd-data** *osddata*

Use object store at *osddata*.

**-osd-journal** *journal*

Journal updates to *journal*.

**-mkfs**

Create an empty object repository. Normally invoked by *mkcephfs(8)*. This also initializes the journal (if one is defined).

**-mkkey**

Generate a new secret key. This is normally used in combination with `--mkfs` as it is more convenient than generating a key by hand with *ceph-authtool(8)*.

**-mkjournal**

Create a new journal file to match an existing object repository. This is useful if the journal device or file is wiped out due to a disk or file system failure.

**-flush-journal**

Flush the journal to permanent store. This runs in the foreground so you know when it's completed. This can be useful if you want to resize the journal or need to otherwise destroy it: this guarantees you won't lose data.

**-get-cluster-fsid**

Print the cluster fsid (uuid) and exit.

**-get-osd-fsid**

Print the OSD's fsid and exit. The OSD's uuid is generated at `-mkfs` time and is thus unique to a particular instantiation of this OSD.

**-get-journal-fsid**

Print the journal's uuid. The journal fsid is set to match the OSD fsid at `-mkfs` time.

**-c** `ceph.conf`, **-conf**=`ceph.conf`

Use *ceph.conf* configuration file instead of the default `/etc/ceph/ceph.conf` for runtime configuration options.

**-m** `monaddress[:port]`

Connect to specified monitor (instead of looking through `ceph.conf`).

## Availability

**ceph-osd** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph*(8), *ceph-mds*(8), *ceph-mon*(8), *ceph-authtool*(8)

## 14.2.11 ceph-rbdnamer – udev helper to name RBD devices

### Synopsis

**ceph-rbdnamer** *num*

### Description

**ceph-rbdnamer** prints the pool and image name for the given RBD devices to stdout. It is used by *udev* (using a rule like the one below) to set up a device symlink.

```
KERNEL=="rbd[0-9]*", PROGRAM="/usr/bin/ceph-rbdnamer %n", SYMLINK+="rbd/%c{1}/%c{2}"
```

### Availability

**ceph-rbdnamer** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*rbd(8)*, *ceph(8)*

### 14.2.12 ceph-run – restart daemon on core dump

#### Synopsis

**ceph-run** *command* ...

#### Description

**ceph-run** is a simple wrapper that will restart a daemon if it exits with a signal indicating it crashed and possibly core dumped (that is, signals 3, 4, 5, 6, 8, or 11).

The command should run the daemon in the foreground. For Ceph daemons, that means the `-f` option.

#### Options

None

#### Availability

**ceph-run** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph(8)*, *ceph-mon(8)*, *ceph-mds(8)*, *ceph-osd(8)*

### 14.2.13 ceph-syn – ceph synthetic workload generator

#### Synopsis

**ceph-syn** [ `-m monaddr:port` ] `--syn command` ...

#### Description

**ceph-syn** is a simple synthetic workload generator for the Ceph distributed file system. It uses the userspace client library to generate simple workloads against a currently running file system. The file system need not be mounted via *ceph-fuse(8)* or the kernel client.

One or more `--syn` command arguments specify the particular workload, as documented below.

## Options

- d**  
Detach from console and daemonize after startup.
- c** *ceph.conf*, **-conf**=*ceph.conf*  
Use *ceph.conf* configuration file instead of the default */etc/ceph/ceph.conf* to determine monitor addresses during startup.
- m** *monaddress[:port]*  
Connect to specified monitor (instead of looking through *ceph.conf*).
- num\_client** *num*  
Run *num* different clients, each in a separate thread.
- syn** *workloadspec*  
Run the given workload. May be specified as many times as needed. Workloads will normally run sequentially.

## Workloads

Each workload should be preceded by **--syn** on the command line. This is not a complete list.

**mknap** *path snapname* Create a snapshot called *snapname* on *path*.

**rmsnap** *path snapname* Delete snapshot called *snapname* on *path*.

**rmfile** *path* Delete/unlink *path*.

**writefile** *sizeinmb blocksize* Create a file, named after our client id, that is *sizeinmb* MB by writing *blocksize* chunks.

**readfile** *sizeinmb blocksize* Read file, named after our client id, that is *sizeinmb* MB by writing *blocksize* chunks.

**rw** *sizeinmb blocksize* Write file, then read it back, as above.

**makedirs** *numsubdirs numfiles depth* Create a hierarchy of directories that is *depth* levels deep. Give each directory *numsubdirs* subdirectories and *numfiles* files.

**walk** Recursively walk the file system (like find).

## Availability

**ceph-syn** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph*(8), *ceph-fuse*(8)

## 14.2.14 cephfs – ceph file system options utility

### Synopsis

**cephfs** [ *path command options* ]

## Description

**cephfs** is a control utility for accessing and manipulating file layout and location data in the Ceph distributed file system.

Choose one of the following three commands:

- `show_layout` View the layout information on a file or directory
- `set_layout` Set the layout information on a file or directory
- `show_location` View the location information on a file

## Options

Your applicable options differ depending on whether you are setting or viewing layout/location.

### Viewing options:

- l** `-offset`  
Specify an offset for which to retrieve location data

### Setting options:

- u** `-stripe_unit`  
Set the size of each stripe
- c** `-stripe_count`  
Set the number of stripes per object
- s** `-object_size`  
Set the size of the objects to stripe across
- p** `-pool`  
Set the pool (by numeric value, not name!) to use
- o** `-osd`  
Set the preferred OSD to use as the primary

## Limitations

When setting layout data, the specified stripe unit and stripe count must multiply to the size of an object. Any parameters you don't set explicitly are left at the system defaults.

Obviously setting the layout of a file and a directory means different things. Setting the layout of a file specifies exactly how to place the individual file. This must be done before writing *any* data to it. Truncating a file does not allow you to change the layout either.

Setting the layout of a directory sets the “default layout”, which is used to set the file layouts on any files subsequently created in the directory (or any subdirectory). Pre-existing files do not have their layouts changed.

You'll notice that the layout information allows you to specify a preferred OSD for placement. This is allowed but is not recommended since it can dramatically unbalance your storage cluster's space utilization.

## Availability

**cephfs** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph*(8)

## 14.2.15 crushtool – CRUSH map manipulation tool

### Synopsis

```
crushtool ( -d map | -c map.txt | -build numosds layer1 ... ) [ -o outfile [ -clobber ] ]
```

### Description

**crushtool** is a utility that lets you create, compile, and decompile CRUSH map files.

CRUSH is a pseudo-random data distribution algorithm that efficiently maps input values (typically data objects) across a heterogeneous, hierarchically structured device map. The algorithm was originally described in detail in the following paper (although it has evolved some since then):

<http://www.ssrc.ucsc.edu/Papers/weil-sc06.pdf>

The tool has three modes of operation.

- c** *map.txt*  
will compile a plaintext *map.txt* into a binary map file.
- d** *map*  
will take the compiled map and decompile it into a plaintext source file, suitable for editing.
- build** *numosds layer1 ...*  
will create a relatively generic map with the given layer structure. See below for examples.

### Options

- o** *outfile*  
will specify the output file.
- clobber**  
will allow the tool to overwrite an existing outfile (it will normally refuse).

### Building a map

The build mode will generate relatively generic hierarchical maps. The first argument simply specifies the number of devices (leaves) in the CRUSH hierarchy. Each layer describes how the layer (or raw devices) preceding it should be grouped.

Each layer consists of:

```
name ( uniform | list | tree | straw ) size
```

The first element is the name for the elements in the layer (e.g. “rack”). Each element’s name will be append a number to the provided name.

The second component is the type of CRUSH bucket.

The third component is the maximum size of the bucket. If the size is 0, a single bucket will be generated that includes everything in the preceding layer.

### Example

Suppose we have 128 devices, each grouped into shelves with 4 devices each, and 8 shelves per rack. We could create a three level hierarchy with:

```
crushtool --build 128 shelf uniform 4 rack straw 8 root straw 0 -o map
```

To adjust the default (generic) mapping rules, we can run:

```
# decompile
crushtool -d map -o map.txt

# edit
vi map.txt

# recompile
crushtool -c map.txt -o map
```

### Availability

**crushtool** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

### See also

*ceph(8)*, *osdmapprool(8)*, *mkcephfs(8)*

## 14.2.16 librados-config – display information about librados

### Synopsis

```
librados-config [ -version ] [ -vernum ]
```

### Description

**librados-config** is a utility that displays information about the installed `librados`.

### Options

```
-version
    Display librados version

-vernum
    Display the librados version code
```

## Availability

**librados-config** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph(8)*, *rados(8)*

## 14.2.17 mkcephfs – create a ceph file system

### Synopsis

**mkcephfs** [ -c *ceph.conf* ] [ -mkbtrfs ] [ -a, -all-hosts [ -k */path/to/admin.keyring* ] ]

### Description

**mkcephfs** is used to create an empty Ceph file system, possibly spanning multiple hosts. The *ceph.conf* file describes the composition of the entire Ceph cluster, including which hosts are participating, which daemons run where, and which paths are used to store file system data or metadata.

The **mkcephfs** tool can be used in two ways. If **-a** is used, it will use *ssh* and *scp* to connect to remote hosts on your behalf and do the setup of the entire cluster. This is the easiest solution, but can also be inconvenient (if you don't have *ssh* to connect without prompting for passwords) or slow (if you have a large cluster).

Alternatively, you can run each setup phase manually. First, you need to prepare a monmap that will be shared by each node:

```
# prepare
master# mkdir /tmp/foo
master# mkcephfs -c /etc/ceph/ceph.conf \
    --prepare-monmap -d /tmp/foo
```

Share the */tmp/foo* directory with other nodes in whatever way is convenient for you. On each OSD and MDS node:

```
osdnode# mkcephfs --init-local-daemons osd -d /tmp/foo
mdsnode# mkcephfs --init-local-daemons mds -d /tmp/foo
```

Collect the contents of the */tmp/foo* directories back onto a single node, and then:

```
master# mkcephfs --prepare-mon -d /tmp/foo
```

Finally, distribute */tmp/foo* to all monitor nodes and, on each of those nodes:

```
monnode# mkcephfs --init-local-daemons mon -d /tmp/foo
```

### Options

#### **-a, -allhosts**

Performs the necessary initialization steps on all hosts in the cluster, executing commands via *SSH*.

#### **-c ceph.conf, -conf=ceph.conf**

Use the given *conf* file instead of the default */etc/ceph/ceph.conf*.



**-k** /path/to/keyring

When **-a** is used, we can specify a location to copy the client.admin keyring, which is used to administer the cluster. The default is `/etc/ceph/keyring` (or whatever is specified in the config file).

**-mkbtrfs**

Create and mount the any btrfs file systems specified in the `ceph.conf` for OSD data storage using `mkfs.btrfs`. The “btrfs devs” and (if it differs from “osd data”) “btrfs path” options must be defined.

**NOTE** Btrfs is still considered experimental. This option can ease some configuration pain, but is the use of btrfs is not required when `osd data` directories are mounted manually by the administrator.

**NOTE** This option is deprecated and will be removed in a future release.

**-no-copy-conf**

By default, `mkcephfs` with **-a** will copy the new configuration to `/etc/ceph/ceph.conf` on each node in the cluster. This option disables that behavior.

## Subcommands

The sub-commands performed during cluster setup can be run individually with

**-prepare-monmap** -d dir -c ceph.conf

Create an initial monmap with a random fsid/uuid and store it and the `ceph.conf` in dir.

**-init-local-daemons** type -d dir

Initialize any daemons of type type on the local host using the monmap in dir. For types `osd` and `mds`, the resulting authentication keys will be placed in dir. For type `mon`, the initial data files generated by `-prepare-mon` (below) are expected in dir.

**-prepare-mon** -d dir

Prepare the initial monitor data based on the monmap, OSD, and MDS authentication keys collected in dir, and put the result in dir.

## Availability

**mkcephfs** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph(8)*, *monmaptool(8)*, *osdmaptool(8)*, *crushtool(8)*

## 14.2.18 monmaptool – ceph monitor cluster map manipulation tool

### Synopsis

**monmaptool** mapfilename [ **-clobber** ] [ **-print** ] [ **-create** ] [ **-add ip:port ...** ] [ **-rm ip:port ...** ]

### Description

**monmaptool** is a utility to create, view, and modify a monitor cluster map for the Ceph distributed file system. The monitor map specifies the only fixed addresses in the Ceph distributed system. All other daemons bind to arbitrary addresses and register themselves with the monitors.

When creating a map with `--create`, a new monitor map with a new, random UUID will be created. It should be followed by one or more monitor addresses.

The default Ceph monitor port is 6789.

### Options

#### **`--print`**

will print a plaintext dump of the map, after any modifications are made.

#### **`--clobber`**

will allow `monmaptool` to overwrite `mapfilename` if changes are made.

#### **`--create`**

will create a new monitor map with a new UUID (and with it, a new, empty Ceph file system).

#### **`--generate`**

generate a new monmap based on the values on the command line or specified in the ceph configuration. This is, in order of preference,

1. `--monmap filename` to specify a monmap to load
2. `--mon-host 'host1,ip2'` to specify a list of hosts or ip addresses
3. `[mon.foo]` sections containing `mon addr` settings in the config

#### **`--filter-initial-members`**

filter the initial monmap by applying the `mon initial members` setting. Monitors not present in that list will be removed, and initial members not present in the map will be added with dummy addresses.

#### **`--add name ip:port`**

will add a monitor with the specified `ip:port` to the map.

#### **`--rm name`**

will remove the monitor with the specified `ip:port` from the map.

#### **`--fsid uuid`**

will set the `fsid` to the given `uuid`. If not specified with `--create`, a random `fsid` will be generated.

### Example

To create a new map with three monitors (for a fresh Ceph file system):

```
monmaptool --create --add mon.a 192.168.0.10:6789 --add mon.b 192.168.0.11:6789 \
--add mon.c 192.168.0.12:6789 --clobber monmap
```

To display the contents of the map:

```
monmaptool --print onmap
```

To replace one monitor:

```
monmaptool --rm mon.a --add mon.a 192.168.0.9:6789 --clobber monmap
```

### Availability

**`monmaptool`** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph*(8), *crushtool*(8), *mkcephfs*(8)

## 14.2.19 mount.ceph – mount a ceph file system

### Synopsis

**mount.ceph** *monaddr1[,monaddr2,...]:[/subdir] dir* [ -o *options* ]

### Description

**mount.ceph** is a simple helper for mounting the Ceph file system on a Linux host. It serves to resolve monitor hostname(s) into IP addresses and read authentication keys from disk; the Linux kernel client component does most of the real work. In fact, it is possible to mount a non-authenticated Ceph file system without **mount.ceph** by specifying monitor address(es) by IP:

```
mount -t ceph 1.2.3.4:/ mountpoint
```

Each monitor address *monaddr* takes the form *host[:port]*. If the port is not specified, the Ceph default of 6789 is assumed.

Multiple monitor addresses can be separated by commas. Only one responsible monitor is needed to successfully mount; the client will learn about all monitors from any responsive monitor. However, it is a good idea to specify more than one in case one happens to be down at the time of mount.

A subdirectory *subdir* may be specified if a subset of the file system is to be mounted.

Mount helper application conventions dictate that the first two options are device to be mounted and destination path. Options must be passed only after these fixed arguments.

### Options

**wsiz** *int*, max write size. Default: none (writeback uses smaller of *wsiz* and stripe unit)

**rsiz** *int* (bytes), max readahead, multiple of 1024, Default: 524288 (512\*1024)

**osdtimeout** *int* (seconds), Default: 60

**osdkeepalivetimeout** *int*, Default: 5

**mount\_timeout** *int* (seconds), Default: 60

**osd\_idle\_ttl** *int* (seconds), Default: 60

**caps\_wanted\_delay\_min** *int*, cap release delay, Default: 5

**caps\_wanted\_delay\_max** *int*, cap release delay, Default: 60

**cap\_release\_safety** *int*, Default: calculated

**readdir\_max\_entries** *int*, Default: 1024

**readdir\_max\_bytes** *int*, Default: 524288 (512\*1024)

**write\_congestion\_kb** *int* (kb), max writeback in flight. scale with available memory. Default: calculated from available memory

**snapdirname** *string*, set the name of the hidden snapdir. Default: .snap

**name** RADOS user to authenticate as when using cephx. Default: guest

**secret** secret key for use with cephx. This option is insecure because it exposes the secret on the command line. To avoid this, use the **secretfile** option.

**secretfile** path to file containing the secret key to use with cephx

**ip** my ip

**noshare** create a new client instance, instead of sharing an existing instance of a client mounting the same cluster

**dirstat** funky *cat dirname* for stats, Default: off

**nodirstat** no funky *cat dirname* for stats

**rbytes** Report the recursive size of the directory contents for *st\_size* on directories. Default: on

**norbytes** Do not report the recursive size of the directory contents for *st\_size* on directories.

**nocrc** no data crc on writes

**noasyncreaddir** no dcache readdir

## Examples

Mount the full file system:

```
mount.ceph monhost:/ /mnt/foo
```

If there are multiple monitors:

```
mount.ceph monhost1,monhost2,monhost3:/ /mnt/foo
```

If *ceph-mon*(8) is running on a non-standard port:

```
mount.ceph monhost1:7000,monhost2:7000,monhost3:7000:/ /mnt/foo
```

To mount only part of the namespace:

```
mount.ceph monhost1:/some/small/thing /mnt/thing
```

Assuming *mount.ceph*(8) is installed properly, it should be automatically invoked by *mount*(8) like so:

```
mount -t ceph monhost:/ /mnt/foo
```

## Availability

**mount.ceph** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph-fuse*(8), *ceph*(8)

## 14.2.20 osdmapprool – ceph osd cluster map manipulation tool

### Synopsis

**osdmapprool** *mapfilename* [-print] [--createsimple *numosd* [-pgbits *bitsperosd* ] ] [-clobber]

## Description

**osdmapprootool** is a utility that lets you create, view, and manipulate OSD cluster maps from the Ceph distributed file system. Notably, it lets you extract the embedded CRUSH map or import a new CRUSH map.

## Options

### **-print**

will simply make the tool print a plaintext dump of the map, after any modifications are made.

### **-clobber**

will allow osdmapprootool to overwrite mapfilename if changes are made.

### **-import-crush** mapfile

will load the CRUSH map from mapfile and embed it in the OSD map.

### **-export-crush** mapfile

will extract the CRUSH map from the OSD map and write it to mapfile.

### **-createsimple** numosd [-pgbits bitsperosd]

will create a relatively generic OSD map with the numosd devices. If -pgbits is specified, the initial placement group counts will be set with bitsperosd bits per OSD. That is, the pg\_num map attribute will be set to numosd shifted by bitsperosd.

## Example

To create a simple map with 16 devices:

```
osdmapprootool --createsimple 16 osdmap --clobber
```

To view the result:

```
osdmapprootool --print osdmap
```

## Availability

**osdmapprootool** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph*(8), *crushtool*(8), *mkcephfs*(8)

## 14.2.21 rados – rados object storage utility

### Synopsis

```
rados [ -m monaddr ] [ mkpool | rmpool foo ] [ -p | -pool pool ] [ -s | -snap snap ] [ -i infile ] [ -o outfile ] command ...
```

### Description

**rados** is a utility for interacting with a Ceph object storage cluster (RADOS), part of the Ceph distributed file system.

## Options

- p** *pool*, **-pool** *pool*  
Interact with the given pool. Required by most commands.
- s** *snap*, **-snap** *snap*  
Read from the given pool snapshot. Valid for all pool-specific read operations.
- i** *infile*  
will specify an input file to be passed along as a payload with the command to the monitor cluster. This is only used for specific monitor commands.
- o** *outfile*  
will write any payload returned by the monitor cluster with its reply to outfile. Only specific monitor commands (e.g. `osd getmap`) return a payload.
- c** *ceph.conf*, **-conf**=*ceph.conf*  
Use *ceph.conf* configuration file instead of the default `/etc/ceph/ceph.conf` to determine monitor addresses during startup.
- m** *monaddress[:port]*  
Connect to specified monitor (instead of looking through *ceph.conf*).

## Global commands

- lspools** List object pools
- df** Show utilization statistics, including disk usage (bytes) and object counts, over the entire system and broken down by pool.
- mkpool** *foo* Create a pool with name *foo*.
- rmpool** *foo* Delete the pool *foo* (and all its data)

## Pool specific commands

- get** *name outfile* Read object name from the cluster and write it to outfile.
- put** *name infile* Write object name to the cluster with contents from infile.
- rm** *name* Remove object name.
- ls** *outfile* List objects in given pool and write to outfile.
- lssnap** List snapshots for given pool.
- mksnap** *foo* Create pool snapshot named *foo*.
- rmsnap** *foo* Remove pool snapshot names *foo*.
- bench** *seconds mode* [ **-b** *objsize* ] [ **-t** *threads* ] Benchmark for seconds. The mode can be write or read. The default object size is 4 KB, and the default number of simulated threads (parallel writes) is 16.
- listomapkeys** *name* List all the keys stored in the object map of object name.
- listomapvals** *name* List all key/value pairs stored in the object map of object name. The values are dumped in hexadecimal.
- getomapval** *name key* Dump the hexadecimal value of key in the object map of object name.
- setomapval** *name key value* Set the value of key in the object map of object name.

**rmomapkey** *name key* Remove key from the object map of object name.

**getomapheader** *name* Dump the hexadecimal value of the object map header of object name.

**setomapheader** *name value* Set the value of the object map header of object name.

## Examples

To view cluster utilization:

```
rados df
```

To get a list object in pool foo sent to stdout:

```
rados -p foo ls -
```

To write an object:

```
rados -p foo put myobject blah.txt
```

To create a snapshot:

```
rados -p foo mksnap mysnap
```

To delete the object:

```
rados -p foo rm myobject
```

To read a previously snapshotted version of an object:

```
rados -p foo -s mysnap get myobject blah.txt.old
```

## Availability

**rados** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph*(8)

## 14.2.22 radosgw – rados REST gateway

### Synopsis

**radosgw**

### Description

**radosgw** is an HTTP REST gateway for the RADOS object store, a part of the Ceph distributed storage system. It is implemented as a FastCGI module using libfcgi, and can be used in conjunction with any FastCGI capable web server.

## Options

- c** *ceph.conf*, **-conf**=*ceph.conf*  
Use *ceph.conf* configuration file instead of the default */etc/ceph/ceph.conf* to determine monitor addresses during startup.
- m** *monaddress[:port]*  
Connect to specified monitor (instead of looking through *ceph.conf*).
- rgw-socket-path**=*path*  
Specify a unix domain socket path.

## Configuration

Currently it's the easiest to use the RADOS Gateway with Apache and *mod\_fastcgi*:

```
FastCgiExternalServer /var/www/s3gw.fcgi -socket /tmp/radosgw.sock
```

```
<VirtualHost *:80>
  ServerName rgw.example1.com
  ServerAlias rgw
  ServerAdmin webmaster@example1.com
  DocumentRoot /var/www

  RewriteEngine On
  RewriteRule ^/([a-zA-Z0-9-_.]*) ([/]?.*) /s3gw.fcgi?page=$1&params=$2%{QUERY_STRING} [E=HTTP_AUTHO

  <IfModule mod_fastcgi.c>
    <Directory /var/www>
      Options +ExecCGI
      AllowOverride All
      SetHandler fastcgi-script
      Order allow,deny
      Allow from all
      AuthBasicAuthoritative Off
    </Directory>
  </IfModule>

  AllowEncodedSlashes On
  ServerSignature Off
</VirtualHost>
```

And the corresponding *radosgw* script (*/var/www/s3gw.fcgi*):

```
#!/bin/sh
exec /usr/bin/radosgw -c /etc/ceph/ceph.conf -n client.radosgw.gateway
```

The *radosgw* daemon is a standalone process which needs a configuration section in the *ceph.conf*. The section name should start with 'client.radosgw.' as specified in */etc/init.d/radosgw*:

```
[client.radosgw.gateway]
  host = gateway
  keyring = /etc/ceph/keyring.radosgw.gateway
  rgw socket path = /tmp/radosgw.sock
```

You will also have to generate a key for the *radosgw* to use for authentication with the cluster:

```
ceph-authtool -C -n client.radosgw.gateway --gen-key /etc/ceph/keyring.radosgw.gateway
ceph-authtool -n client.radosgw.gateway --cap mon 'allow r' --cap osd 'allow rwx' /etc/ceph/keyring.
```



And add the key to the auth entries:

```
ceph auth add client.radosgw.gateway --in-file=keyring.radosgw.gateway
```

Now you can start Apache and the radosgw daemon:

```
/etc/init.d/apache2 start
/etc/init.d/radosgw start
```

## Usage Logging

The **radosgw** maintains an asynchronous usage log. It accumulates statistics about user operations and flushes it periodically. The logs can be accessed and managed through **radosgw-admin**.

The information that is being logged contains total data transfer, total operations, and total successful operations. The data is being accounted in an hourly resolution under the bucket owner, unless the operation was done on the service (e.g., when listing a bucket) in which case it is accounted under the operating user.

Following is an example configuration:

```
[client.radosgw.gateway]
    rgw enable usage log = true
    rgw usage log tick interval = 30
    rgw usage log flush threshold = 1024
    rgw usage max shards = 32
    rgw usage max user shards = 1
```

The total number of shards determines how many total objects hold the usage log information. The per-user number of shards specify how many objects hold usage information for a single user. The tick interval configures the number of seconds between log flushes, and the flush threshold specify how many entries can be kept before resorting to synchronous flush.

## Availability

**radosgw** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph(8) radosgw-admin(8)*

### 14.2.23 radosgw-admin – rados REST gateway user administration utility

#### Synopsis

**radosgw-admin** *command* [ *options* ... ]

#### Description

**radosgw-admin** is a RADOS gateway user administration utility. It allows creating and modifying users.

## Commands

*command* can be one of the following options:

**user create** Create a new user

**user modify** Modify a user

**user info** Display information of a user, and any potentially available subusers and keys

**user rm** Remove a user

**subuser create** Create a new subuser (primarily useful for clients using the Swift API)

**subuser modify** Modify a subuser

**subuser rm** Remove a subuser

**bucket list** List all buckets

**bucket unlink** Remove a bucket

**key create** Create an access key

**key rm** Remove an access key

**policy** Display bucket/object policy

**log show** Show the log of a bucket (with a specified date)

**usage show** Show the usage information (with optional user and date range)

**usage trim** Trim usage information (with optional user and date range)

## Options

**-c** *ceph.conf*, **-conf**=*ceph.conf*

Use *ceph.conf* configuration file instead of the default */etc/ceph/ceph.conf* to determine monitor addresses during startup.

**-m** *monaddress[:port]*

Connect to specified monitor (instead of looking through *ceph.conf*).

**-uid**=*uid*

The radosgw user ID.

**-secret**=*secret*

The secret associated with a given key.

**-display-name**=*name*

Configure the display name of the user.

**-email**=*email*

The e-mail address of the user

**-bucket**=*bucket*

Specify the bucket name.

**-object**=*object*

Specify the object name.

**-date**=*yyyy-mm-dd*

The date needed for some commands

**-start-date**=yyyy-mm-dd  
The start date needed for some commands

**-end-date**=yyyy-mm-dd  
The end date needed for some commands

**-auth-uid**=auid  
The librados auid

## Examples

Generate a new user:

```
$ radosgw-admin user create --display-name="johnny rotten" --uid=johnny
{ "user_id": "johnny",
  "rados_uid": 0,
  "display_name": "johnny rotten",
  "email": "",
  "suspended": 0,
  "subusers": [],
  "keys": [
    { "user": "johnny",
      "access_key": "TCICW53D9BQ2VGC46I44",
      "secret_key": "tfm9aHMI8X76L3UdgE+ZQaJag1vJQmE6HDb5Lbrz"}],
  "swift_keys": []}
```

Remove a user:

```
$ radosgw-admin user rm --uid=johnny
```

Remove a bucket:

```
$ radosgw-admin bucket unlink --bucket=foo
```

Show the logs of a bucket from April 1st, 2012:

```
$ radosgw-admin log show --bucket=foo --date=2012=04-01
```

Show usage information for user from March 1st to (but not including) April 1st, 2012:

```
$ radosgw-admin usage show --uid=johnny \
  --start-date=2012-03-01 --end-date=2012-04-01
```

Show only summary of usage information for all users:

```
$ radosgw-admin usage show --show-log-entries=false
```

Trim usage information for user until March 1st, 2012:

```
$ radosgw-admin usage trim --uid=johnny --end-date=2012-04-01
```

## Availability

**radosgw-admin** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

## See also

*ceph*(8)

### 14.2.24 rbd – manage rados block device (RBD) images

#### Synopsis

**rbd** [ **-c** *ceph.conf* ] [ **-m** *monaddr* ] [ **-p** | **-pool** *pool* ] [ **-size** *size* ] [ **-order** *bits* ] [ *command ...* ]

#### Description

**rbd** is a utility for manipulating rados block device (RBD) images, used by the Linux rbd driver and the rbd storage driver for Qemu/KVM. RBD images are simple block devices that are striped over objects and stored in a RADOS object store. The size of the objects the image is striped over must be a power of two.

#### Options

- c** *ceph.conf*, **-conf** *ceph.conf*  
Use *ceph.conf* configuration file instead of the default */etc/ceph/ceph.conf* to determine monitor addresses during startup.
- m** *monaddress[:port]*  
Connect to specified monitor (instead of looking through *ceph.conf*).
- p** *pool*, **-pool** *pool*  
Interact with the given pool. Required by most commands.

#### Parameters

- size** *size-in-mb*  
Specifies the size (in megabytes) of the new rbd image.
- order** *bits*  
Specifies the object size expressed as a number of bits, such that the object size is  $1 \ll \text{order}$ . The default is 22 (4 MB).
- snap** *snap*  
Specifies the snapshot name for the specific operation.
- user** *username*  
Specifies the username to use with the map command.
- secret** *filename*  
Specifies a file containing the secret to use with the map command.

#### Commands

- ls** [*pool-name*] Will list all rbd images listed in the *rbd\_directory* object.
- info** [*image-name*] Will dump information (such as size and order) about a specific rbd image.
- create** [*image-name*] Will create a new rbd image. You must also specify the size via **-size**.

**clone** [*parent-snapname*] [*image-name*] Will create a clone (copy-on-write child) of the parent snapshot. Size and object order will be identical to parent image unless specified.

**resize** [*image-name*] Resizes rbd image. The size parameter also needs to be specified.

**rm** [*image-name*] Deletes an rbd image (including all data blocks). If the image has snapshots, this fails and nothing is deleted.

**export** [*image-name*] [*dest-path*] Exports image to dest path.

**import** [*path*] [*dest-image*] Creates a new image and imports its data from path.

**cp** [*src-image*] [*dest-image*] Copies the content of a src-image into the newly created dest-image.

**mv** [*src-image*] [*dest-image*] Renames an image. Note: rename across pools is not supported.

**snap ls** [*image-name*] Dumps the list of snapshots inside a specific image.

**snap create** [*image-name*] Creates a new snapshot. Requires the snapshot name parameter specified.

**snap rollback** [*image-name*] Rollback image content to snapshot. This will iterate through the entire blocks array and update the data head content to the snapshotted version.

**snap rm** [*image-name*] Removes the specified snapshot.

**snap purge** [*image-name*] Removes all snapshots from an image.

**map** [*image-name*] Maps the specified image to a block device via the rbd kernel module.

**unmap** [*device-path*] Unmaps the block device that was mapped via the rbd kernel module.

**showmapped** Show the rbd images that are mapped via the rbd kernel module.

## Image name

In addition to using the `-pool` and the `-snap` options, the image name can include both the pool name and the snapshot name. The image name format is as follows:

```
[pool/]image-name[@snap]
```

Thus an image name that contains a slash character (`/`) requires specifying the pool name explicitly.

## Examples

To create a new rbd image that is 100 GB:

```
rbd -p mypool create myimage --size 102400
```

or alternatively:

```
rbd create mypool/myimage --size 102400
```

To use a non-default object size (8 MB):

```
rbd create mypool/myimage --size 102400 --order 23
```

To delete an rbd image (be careful!):

```
rbd rm mypool/myimage
```

To create a new snapshot:

```
rbd snap create mypool/myimage@mysnap
```

To create a copy-on-write clone of a snapshot:

```
rbd clone myimage@mysnap cloneimage
```

To delete a snapshot:

```
rbd snap rm mypool/myimage@mysnap
```

To map an image via the kernel with cephx enabled:

```
rbd map myimage --user admin --secret secretfile
```

To unmap an image:

```
rbd unmap /dev/rbd0
```

### Availability

**rbd** is part of the Ceph distributed file system. Please refer to the Ceph documentation at <http://ceph.com/docs> for more information.

### See also

*ceph*(8), *rados*(8)

# ARCHITECTURE OF CEPH

Ceph is a distributed network storage and file system with distributed metadata management and POSIX semantics.

RADOS is a reliable object store, used by Ceph, but also directly accessible.

`radosgw` is an S3-compatible RESTful HTTP service for object storage, using RADOS storage.

RBD is a Linux kernel feature that exposes RADOS storage as a block device. Qemu/KVM also has a direct RBD client, that avoids the kernel overhead.

## 15.1 Monitor cluster

`ceph-mon` is a lightweight daemon that provides a consensus for distributed decisionmaking in a Ceph/RADOS cluster.

It also is the initial point of contact for new clients, and will hand out information about the topology of the cluster, such as the `osdmap`.

You normally run 3 `ceph-mon` daemons, on 3 separate physical machines, isolated from each other; for example, in different racks or rows.

You could run just 1 instance, but that means giving up on high availability.

You may use the same hosts for `ceph-mon` and other purposes.

`ceph-mon` processes talk to each other using a [Paxos](#)-style protocol. They discover each other via the `[mon.X] mon addr` fields in `ceph.conf`.

---

### Todo

What about `monmap`? Fact check.

---

Any decision requires the majority of the `ceph-mon` processes to be healthy and communicating with each other. For this reason, you never want an even number of `ceph-mons`; there is no unambiguous majority subgroup for an even number.

---

### Todo

explain `monmap`

---

## 15.2 RADOS

`ceph-osd` is the storage daemon that provides the RADOS service. It uses `ceph-mon` for cluster membership, services object read/write/etc request from clients, and peers with other `ceph-osds` for data replication.

The data model is fairly simple on this level. There are multiple named pools, and within each pool there are named objects, in a flat namespace (no directories). Each object has both data and metadata.

The data for an object is a single, potentially big, series of bytes. Additionally, the series may be sparse, it may have holes that contain binary zeros, and take up no actual storage.

The metadata is an unordered set of key-value pairs. It's semantics are completely up to the client; for example, the Ceph filesystem uses metadata to store file owner etc.

---

### Todo

Verify that metadata is unordered.

---

Underneath, `ceph-osd` stores the data on a local filesystem. We recommend using [Btrfs](#), but any POSIX filesystem that has extended attributes should work.

---

### Todo

write about access control

---

### Todo

explain osdmap

---

### Todo

explain plugins ("classes")

---

## 15.3 Ceph filesystem

The Ceph filesystem service is provided by a daemon called `ceph-mds`. It uses RADOS to store all the filesystem metadata (directories, file ownership, access modes, etc), and directs clients to access RADOS directly for the file contents.

The Ceph filesystem aims for POSIX compatibility, except for a few chosen differences. See [Differences from POSIX](#).

`ceph-mds` can run as a single process, or it can be distributed out to multiple physical machines, either for high availability or for scalability.

For high availability, the extra `ceph-mds` instances can be *standby*, ready to take over the duties of any failed `ceph-mds` that was *active*. This is easy because all the data, including the journal, is stored on RADOS. The transition is triggered automatically by `ceph-mon`.

For scalability, multiple `ceph-mds` instances can be *active*, and they will split the directory tree into subtrees (and shards of a single busy directory), effectively balancing the load amongst all *active* servers.

Combinations of *standby* and *active* etc are possible, for example running 3 *active* `ceph-mds` instances for scaling, and one *standby*.



To control the number of *active* `ceph-mdses`, see *Resizing the metadata cluster*.

---

**Status as of 2011-09:**

Multiple *active* `ceph-mds` operation is stable under normal circumstances, but some failure scenarios may still cause operational issues.

---

**Todo**

document *standby-replay*

---

**Todo**

mds.0 vs mds.alpha etc details

---

## 15.4 radosgw

`radosgw` is a FastCGI service that provides a [RESTful](#) HTTP API to store objects and metadata. It layers on top of RADOS with its own data formats, and maintains it's own user database, authentication, access control, and so on.

## 15.5 Rados Block Device (RBD)

In virtual machine scenarios, RBD is typically used via the `rbd` network storage driver in Qemu/KVM, where the host machine uses `librbd` to provide a block device service to the guest.

Alternatively, as no direct `librbd` support is available in Xen, the Linux kernel can act as the RBD client and provide a real block device on the host machine, that can then be accessed by the virtualization. This is done with the command-line tool `rbd` (see *RBD setup and administration*).

The latter is also useful in non-virtualized scenarios.

Internally, RBD stripes the device image over multiple RADOS objects, each typically located on a separate `ceph-osd`, allowing it to perform better than a single server could.

## 15.6 Client

---

**Todo**

`cephfs`, `ceph-fuse`, `librados`, `libcephfs`, `librbd`

---

**Todo**

Summarize how much Ceph trusts the client, for what parts (security vs reliability).

---

## 15.7 TODO

---

### Todo

Example scenarios Ceph projects are/not suitable for

---

# FREQUENTLY ASKED QUESTIONS

These questions have been frequently asked on the ceph-devel mailing list, the IRC channel, and on the Ceph.com blog.

## 16.1 Is Ceph Production-Quality?

The definition of “production quality” varies depending on who you ask. Because it can mean a lot of different things depending on how you want to use Ceph, we prefer not to think of it as a binary term.

At this point we support the RADOS object store, radosgw, and rbd because we think they are sufficiently stable that we can handle the support workload. There are several organizations running those parts of the system in production. Others wouldn’t dream of doing so at this stage.

The CephFS POSIX-compliant filesystem is functionally-complete and has been evaluated by a large community of users, but has not yet been subjected to extensive, methodical testing.

We can tell you how we test, and what we support, but in the end it’s your judgement that matters most!

## 16.2 How can I add a question to this list?

If you’d like to add a question to this list (hopefully with an accompanying answer!), you can find it in the doc/ directory of our main git repository:

<https://github.com/ceph/ceph/blob/master/doc/faq.rst>

We use Sphinx to manage our documentation, and this page is generated from reStructuredText source. See the section on Building Ceph Documentation for the build procedure.



# ACADEMIC PAPERS

---

## Todo

transfer content from <http://ceph.com/resources/publications/> ?

---



# RELEASE NOTES

## 18.1 v0.48 “argonaut”

### 18.1.1 Upgrading

- This release includes a disk format upgrade. Each ceph-osd daemon, upon startup, will migrate its locally stored data to the new format. This process can take a while (for large object counts, even hours), especially on non-btrfs file systems.
- To keep the cluster available while the upgrade is in progress, we recommend you upgrade a storage node or rack at a time, and wait for the cluster to recover each time. To prevent the cluster from moving data around in response to the OSD daemons being down for minutes or hours, you may want to:

```
ceph osd set noout
```

This will prevent the cluster from marking down OSDs as “out” and re-replicating the data elsewhere. If you do this, be sure to clear the flag when the upgrade is complete:

```
ceph osd unset noout
```

- There is a encoding format change internal to the monitor cluster. The monitor daemons are careful to switch to the new format only when all members of the quorum support it. However, that means that a partial quorum with new code may move to the new format, and a recovering monitor running old code will be unable to join (it will crash). If this occurs, simply upgrading the remaining monitor will resolve the problem.
- The ceph tool’s -s and -w commands from previous versions are incompatible with this version. Upgrade your client tools at the same time you upgrade the monitors if you rely on those commands.
- It is not possible to downgrade from v0.48 to a previous version.

### 18.1.2 Notable changes

- osd: stability improvements
- osd: capability model simplification
- osd: simpler/safer `--mkfs` (no longer removes all files; safe to re-run on active osd)
- osd: potentially buggy FIEMAP behavior disabled by default
- rbd: caching improvements
- rbd: improved instrumentation
- rbd: bug fixes

- radosgw: new, scalable usage logging infrastructure
- radosgw: per-user bucket limits
- mon: streamlined process for setting up authentication keys
- mon: stability improvements
- mon: log message throttling
- doc: improved documentation (ceph, rbd, radosgw, chef, etc.)
- config: new default locations for daemon keyrings
- config: arbitrary variable substitutions
- improved ‘admin socket’ daemon admin interface (`ceph --admin-daemon ...`)
- chef: support for multiple monitor clusters
- upstart: basic support for monitors, mds, radosgw; osd support still a work in progress.

The new default keyring locations mean that when enabling authentication (`auth supported = cephx`), keyring locations do not need to be specified if the keyring file is located inside the daemon’s data directory (`/var/lib/ceph/$type/ceph-$id` by default).

There is also a lot of librbd code in this release that is laying the groundwork for the upcoming layering functionality, but is not actually used. Likewise, the upstart support is still incomplete and not recommended; we will backport that functionality later if it turns out to be non-disruptive.



# APPENDICES

## 19.1 Differences from POSIX

---

### Todo

delete [http://ceph.com/wiki/Differences\\_from\\_POSIX](http://ceph.com/wiki/Differences_from_POSIX)

---

Ceph does have a few places where it diverges from strict POSIX semantics for various reasons:

- Sparse files propagate incorrectly to tools like `df`. They will only use up the required space, but in `df` will increase the “used” space by the full file size. We do this because actually keeping track of the space a large, sparse file uses is very expensive.
- In shared simultaneous writer situations, a write that crosses object boundaries is not necessarily atomic. This means that you could have writer A write “a`l`aa” and writer B write “bb`l`bb” simultaneously (where `l` is the object boundary), and end up with “aa`l`bb” rather than the proper “a`l`aa” or “bb`l`bb”.



# PYTHON MODULE INDEX

## r

rbid, [172](#)